Assessing the Use of Slicing-based Visualizing Techniques on the Understanding of Large Metamodels

Arnaud Blouin, Naouel Moha, Benoit Baudry, Houari Sahraoui, Jean-Marc Jézéquel

Assessing the Use of Slicing-based Visualizing Techniques on the Understanding of Large Metamodels

Arnaud Blouin[a], Naouel Moha[b], Benoit Baudry[c], Houari Sahraoui[d], Jean-Marc Jézéquel[e]

[a]*INSA Rennes, IRISA / Inria, Diverse Team, Rennes, France*
[b]*University of Québec at Montréal, Montréal, Canada*
[c]*Inria, IRISA / Inria, Diverse Team, Rennes, France*
[d]*University of Montréal, GEODES Group, Montréal, Canada*
[e]*University of Rennes 1, IRISA / Inria, Diverse Team, Rennes, France*

Abstract

Context. Metamodels are cornerstones of various metamodeling activities. Such activities consist of, for instance, transforming models into code or comparing metamodels. These activities thus require a good understanding of a metamodel and/or its parts. Current metamodel editing tools are based on standard interactive visualization features, such as physical zooms.
Objective. However, as soon as metamodels become large, navigating through large metamodels becomes a tedious task that hinders their understanding. So, a real need to support metamodel comprehension appears.
Method. In this work we promote the use of model slicing techniques to build interactive visualization tools for metamodels. Model slicing is a model comprehension technique inspired by program slicing. We show how the use of `Kompren`, a domain-specific language for defining model slicers, can ease the development of such interactive visualization features.
Results. We specifically make four main contributions. First, the proposed interactive visualization techniques permit users to focus on metamodel elements of interest, which aims at improving the understandability. Second, these proposed techniques are developed based on model slicing, a model comprehension technique that involves extracting a subset of model elements of interest. Third, we develop a metamodel visualizer, called `Explen`, embedding the proposed interactive visualization techniques. Fourth, we conducted experiments. showing that `Explen` significantly outperforms `EcoreTools`, in terms of time, correctness, and navigation effort, on metamodeling tasks.
Conclusion. The results of the experiments, in favor of `Explen`, show that improving metamodel understanding can be done using slicing-based interactive navigation features.

Keywords: Model-Driven Engineering, Metamodel, Class Diagram, Visualization, Human-Computer Interaction, Model Slicing

1. Introduction

The fundamental idea of Model-Driven Engineering (MDE) is to consider models as first-class entities. A model conforms to a metamodel that describes the concepts and relationships of a given domain. Metamodels, usually represented graphically as class diagrams, are thus cornerstones of various metamodeling activities. Such activities consist of, for instance, transforming models into code, creating editing tools for a metamodel, or comparing metamodels. These activities thus require a good understanding of a metamodel and/or its parts. Understanding metamodels mainly consists of understanding the relations between classes of interest by navigating between them through their inheritance or reference relations. The current mainstream metamodel editors, such as `EcoreTools` provided by the Eclipse Modeling Framework (EMF)[1], however, only offer basic interactive features to navigate through metamodels (physical zoom, scroll bars, *etc.*). Physical zooms are used to change the size of metamodels' elements, scroll bars are used to navigate from one class to another one, and several filters permit hiding classes or relations. Although MDE promotes the separation of concerns that should limit the size of metamodels by decomposing them into small ones, empirical evidence shows that many of them are large. An empirical study we conducted on 3462 well-formed Ecore domain metamodels we gathered from the github platform highlighted that: 82 % of the studied metamodels are composed of a single package, the mandatory root package; 15 %, *i.e.* 508, of these metamodels are composed of 40 classes or more. As soon as metamodels become large, understanding and manipulating metamodels becomes a tedious task using these basic interactive features. For instance, Figure 1 is an overview of the UML metamodel [1] obtained using the physical zoom of `EcoreTools`. Many classes are gathered and reduced so that identifying one class or its relations with other ones becomes awkward. As noticed by Zhao *et al.*, "*while node-link diagrams show nesting structure very clearly, they use screen space inefficiently, and do not scale well to large datasets*" [2].

Email addresses: `ablouin@irisa.fr` (Arnaud Blouin), `moha.naouel@uqam.ca` (Naouel Moha), `bbaudry@inria.fr` (Benoit Baudry), `sahraoui@iro.umontreal.ca` (Houari Sahraoui), `jezequel@irisa.fr` (Jean-Marc Jézéquel)

[1] `http://www.eclipse.org/modeling/emf/`

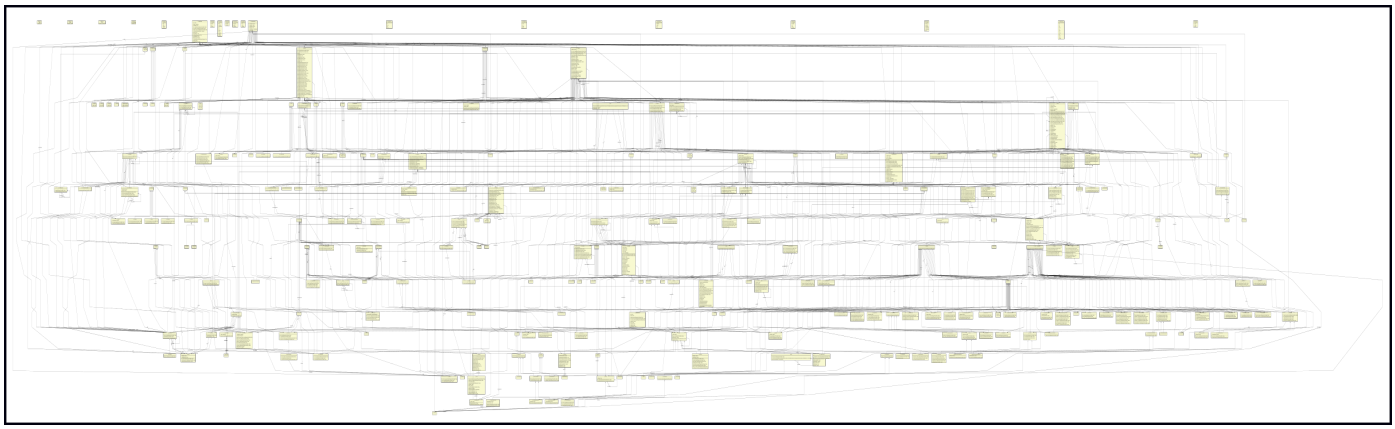

Figure 1: Bird view of the UML metamodel using `EcoreTools` (246 classes and 769 relationships)

When modelers are interested only in a specific part of a metamodel, they may want to focus on it by, for instance, hiding the rest of the metamodel. For instance, for the visualization of a metamodel, a modeler may be interested in semantic relationships between classes such as: the inheritance tree of a given class; the classes linked by a composition reference to a given class. As motivated by Fondement *et al.*, "*by indicating formally the subset of the metamodel that is actually covered, a tool could be made more precise regarding handled model*" [3]. With the current editors, modelers are forced to manually and astutely combine sequences of filtering and navigation primitive operations to rebuild these parts of interest. This manual exploration task may be time-consuming and error-prone. So, a real need to support metamodel comprehension appears.

Visualization techniques are broadly used in software engineering and have proven their usefulness for software comprehension and in particular, interactive visualization that provides meaningful navigation capabilities [4]. Gračanin *et al.* summarized the benefits in terms of comprehension brought by software visualization to different domains such as software evolution, software security, and data mining [5]. Previous works on UML class diagrams highlight the research interest on improving the understanding of class diagrams [6, 7, 8]. These works mainly focus on proposing new algorithms and methods for minimizing relations crossing [9, 10, 11] or guidelines for drawing class diagrams [12, 13]. Other research works proposed to represent class models differently than using class diagrams [14] or in 3D [15, 16, 17]. In this work we focus on metamodeling tasks that modelers perform while handling metamodels. More precisely, we consider how to produce interactive visualization features dedicated to metamodels rather than the rendering of metamodels. We also keep the focus on the class diagram representation promoted and widely-used within the MDE community. We specifically propose four main contributions. First, we propose interactive visualization techniques that permit users to focus on metamodel elements of interest. These techniques aim at improving the understandability of metamodels. Second, these proposed techniques are developed based on model slicing [18, 19]. Model slicing is a model comprehension technique inspired by program slicing [20]. The process of model slicing involves extracting a subset of model elements of interest. We show how the use of `Kompren`, a domain-specific language for defining model slicers [18, 19], can ease the development of such interactive visualization features. Third, we develop a metamodel visualizer, called `Explen`, embedding the proposed interactive visualization techniques. Fourth, we conducted an empirical study to measure the possible benefits, in terms of time, correctness, and navigation effort, when performing metamodeling tasks using `Explen` compared to the mainstream metamodeling tool `EcoreTools`. This study exhibits significant positive results for `Explen` regarding both time (30 % better in favor of `Explen`), correctness (22 % better in favor of `Explen`), and navigation effort (50 % better in favor of `Explen`). This work is the first step towards generalizing the proposal to any kind of models represented with a graphical syntax. It aims at validating the benefits of the proposal on metamodels to then, in future work, consider models in general.

This paper extends our work published at VISSOFT 2014 (*New Ideas or Emerging Results* Track) [21] with an empirical study, an exhaustive study of the related work, and more details explaining the proposed interactive visualization features.

The paper is organized as follows. Section 2 motivates this work by presenting a scenario that highlights the need for integrating interactive visualization features within graphical modeling tools. Section 3 describes how model slicing can be leveraged to develop interactive visualization techniques for the visualization of large metamodels. Section 4 details the experimental design. Section 5 analyses and comments the results of the conducted experiments. The paper ends with the related work in Section 6 and the conclusion in Section 7.

2. Motivating Scenario

We motivate the need for integrating interactive visualization features within graphical modeling tools based on the following common scenario.

Scenario. A modeler has to write a model transformation that generates Java code from UML 2.0 models. The modeler has already a rough idea of the main classes required for the transformation: *Association*, *Class*, *Package*, *Parameter*, *Property*,

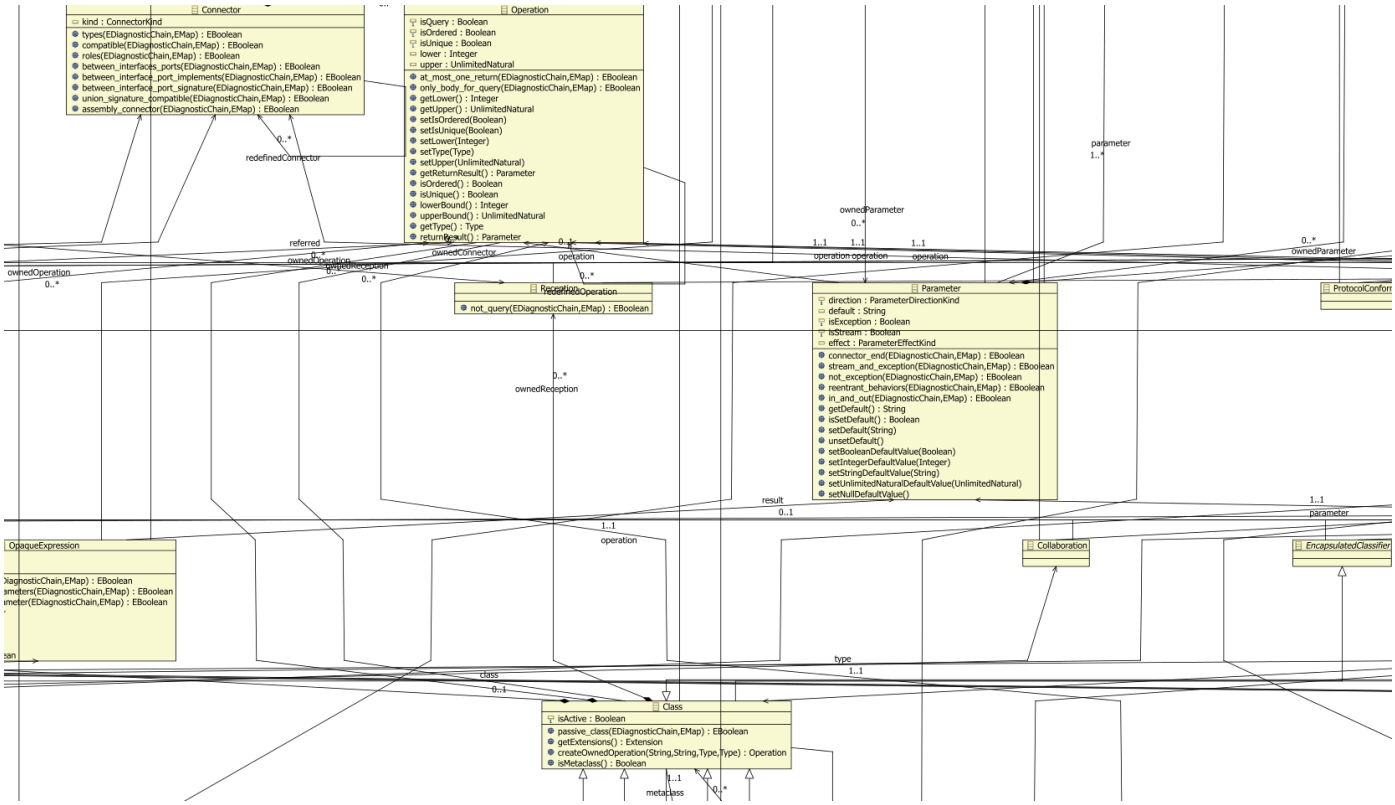

Figure 2: Physical zoom on the UML class *Class*

and *Operation*. However, before writing the transformation, the modeler needs to have a clear and precise understanding of how these classes are organized within the UML metamodel and identify the properties and operations required for the transformation. To acquire this understanding, the modeler visualizes the UML metamodel using, for example, `EcoreTools`[1] of EMF (*Eclipse Modeling Framework*, a broadly used modeling tool). The visualization of the whole UML metamodel is not a great help to her: as illustrated in Figure 1, the UML metamodel is overcrowded because of its 246 classes and 769 relationships (association, composition, inheritance) contained into a single root package. Yet, the modeler navigates and explores the UML metamodel to precisely identify classes and elements (properties and operations) required for her transformation. The following tasks are examples of such a process of navigation and exploration:

Task 1. The modeler uses the physical zoom of the editor to focus on classes related to *Class* as depicted in Figure 2. Since classes directly linked to *Class* do not appear in the zoomed view, and a high number of relationships are tangled, the interest of this view is limited for the modeler. A zoom out of this view will gather too many classes and the view will still be unreadable such as in Figure 1.

Task 2. The modeler explores the inheritance relationships of *Class* to see all elements that may be inherited in *Class*. However, this task is quite difficult because of the tangled relationships and the high number of classes that hinder the visibility. Moreover, the multiple inheritance of several UML classes complicates the navigation within the inheritance tree of *Class*. Using the tree view of the editor's outline, the modeler can find the superclasses of *Class*. For example, the inheritance tree of *Class* is composed of 17 inheritance relations. To get all the inherited properties of *Class*, the modeler needs to navigate through each of these classes. The modeler may also use the documentation (*e.g.* the JavaDoc corresponding to the metamodel under study) to explore the inheritance relationships. This, however, forces the modeler to jump between different representations.

Task 3. The modeler starts to slice the UML metamodel (*i.e.* hide UML elements) to obtain a reduced view of the UML metamodel that contains only relevant classes for her transformation. The modeler performs this slicing using the filtering and navigation capabilities of `EcoreTools`. The navigation capability allows the modeler to restore related elements while the filtering capability allows the modeler to hide elements. For instance, the filter *Hide Selection* enables the modeler to hide all the selected elements; the filters *Hide Inheritance Relations* and *Hide Reference Relations* allow the modeler to hide, respectively, all the inheritance and reference links *in the current entire metamodel* (but not for a selected element). After having sliced manually one by one the 216 classes of the metamodel not concerned by the transformation, the modeler finally obtains a subset of the UML metamodel that contains the 30 relevant classes for her transformation.

3. Leveraging Model Slicing for Developing Metamodel Visualization Techniques

3.1. On the benefits of model slicing to build filtering features

Ideally, the modeler would have preferred to obtain the reduced views in a more straightforward way instead of astutely combining the editor's filtering and navigation capabilities. If the editor had provided a filtering capability to show only classes directly linked to a selected class, Task 1 might have been easier. A similar filtering capability for the inheritance relationships might have eased Task 2. Regarding Task 3, a more complex filtering capability that combines the two previous ones might also have been useful for the modeler. Such interactive visualization features would: permit users to focus on the metamodel elements of interest by hiding the other ones; minimize edge crossings identified as an impacting factor on the cognitive load [22]. So, modelers may want dynamic queries [23] to quickly focus on their interests by eliminating unwanted elements.

Model slicing is a model comprehension technique inspired by program slicing. The process of model slicing involves extracting a subset of model elements that represent a *model slice*. The model slice may vary depending on the intended purpose. For example, when seeking to understand a large metamodel, it may help to extract the sub-part of the metamodel that includes only the dependencies of a particular class. In the following section, we present the notion of model slicing. Then, we introduce the slicing-based interactive visualization features we designed for exploring metamodels. We detail how these features can be developed using model slicing techniques. We also illustrate how the scenario of the previous section can be done using `Explen`.

3.2. Background on Model Slicing

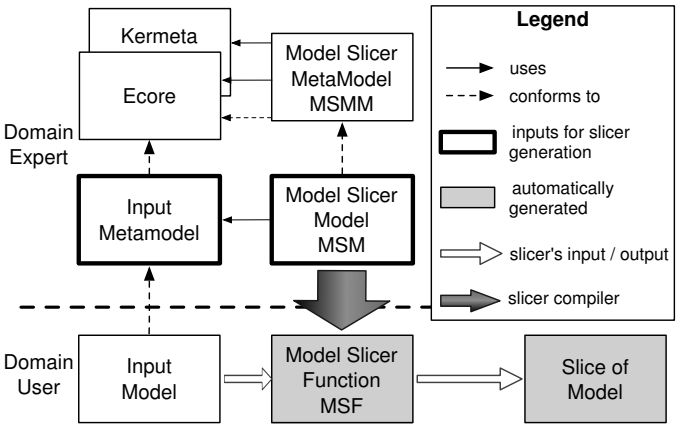

Figure 3: Overview for Modeling Model Slicers with `Kompren`, from [18]

In our previous work we proposed `Kompren`, a domain-specific modeling language to define model slicers for a particular domain [18, 19]. For instance, if a developer wants to slice specific elements of UML models, she may use `Kompren` as described as follows. The developer first uses the `Kompren` language to select the elements of interest from the UML metamodel (captured in the *UML.ecore* model). That will lead to the creation of a `Kompren` model slicer. This model is then compiled into an executable Java program that can slice UML models as specified in the `Kompren` model slicer. More generally, Figure 3 provides an overview of the `Kompren` process to define model slicers. All the concepts and relations of `Kompren` are captured in the *model slicer metamodel* (MSMM at the top of Figure 3). A *model slicer model* (MSM) expressed with `Kompren` refers to a set of classes and relations from the *input metamodel* (in our case, an Ecore model[1]). Instances of the referenced classes and relations will be selected for slicing the *input model*. The MSMM also points to `Kermeta 3`[2] [24], an action language used, in our case, to specify the behavior of a slicer. `Kompren`'s compiler processes an MSM defined for an *input metamodel* and automatically generates a *model slicer function* (MSF). The generated MSF consists of a Java executable program (`Kermeta` programs are compiled into Java). It takes as input an *input model* (instance of the input metamodel) and slicing criteria. Slicing criteria are model elements from the *input model* that provide entry points for extracting a model slice. The execution of a MSF consists of exploring the input model from model elements given as input (the slicing criteria). Each of these elements is visited. Visiting model elements (classes, properties, *etc.*) consists of executing the associated behavior, *i.e.* the corresponding `Kermeta` expression defined by the domain expert in the MSM. Each selected property of the current visited class instance is then explored to recursively explore their target class instance. At then end of the slicing process, a subset of model elements is then obtained.

3.3. `Explen`: a Kompren-Based Metamodel Visualization Tool

As detailed in the previous section, model slicing permits to extract subsets from models. From a visualization perspective, model slicing can be thus used to develop filtering-based visualization techniques, usually called dynamic queries [23]. We used `Kompren` to develop such techniques to visualize metamodels. We develop a metamodel visualization tool, called `Explen`, embedding dedicated visualization techniques we developed using `Kompren`. In this section, we first detail the development process for developing interactive visualization features based on Kompren. We then introduce `Explen` and its visualization techniques dedicated to metamodels.

3.3.1. Development Process

The slicing-based interactive visualization techniques embedded in `Explen` have been developed using `Kompren`. The use of `Kompren` within `Explen` is depicted in Figure 4 and can be applied to any Java metamodel visualizer. Developers defined a `Kompren` model slicer dedicated to slice Ecore metamodels (a language for defining metamodels, depicted in Figure 5). Listing 1 details the code of the developed model slicer, called *MetamodelSlicer* (line 1). Lines 2 and 3 specifies the input metamodel to consider. Line 4 defines that this slicer will take as input instances of the Ecore class *EClass*. The rest of the code and the Ecore metamodel are explained throughout the next section that

[2] `http://www.kermeta.org/`

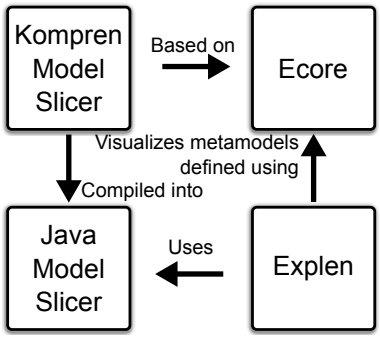

Figure 4: The use of Kompren within Explen

introduces the interactive visualization features. The Kompren model slicer is then compiled into a Java library to be integrated into Explen. This integration consists of binding the user interface of Explen to the model slicer. For instance with Explen, Figure 6 depicts the menu that appears when right-clicking on a class. The widgets of this menu permit to parameterize and call the Kompren model slicer. Once a slicing process is launched, the user interface can be notified about the metamodel elements to hide or show. Explen[3] provides a 2D view of the metamodel under study (see Figure 11). Explen has been developed in Java using the Malai architectural design pattern dedicated to the development of highly interactive systems [25, 26].

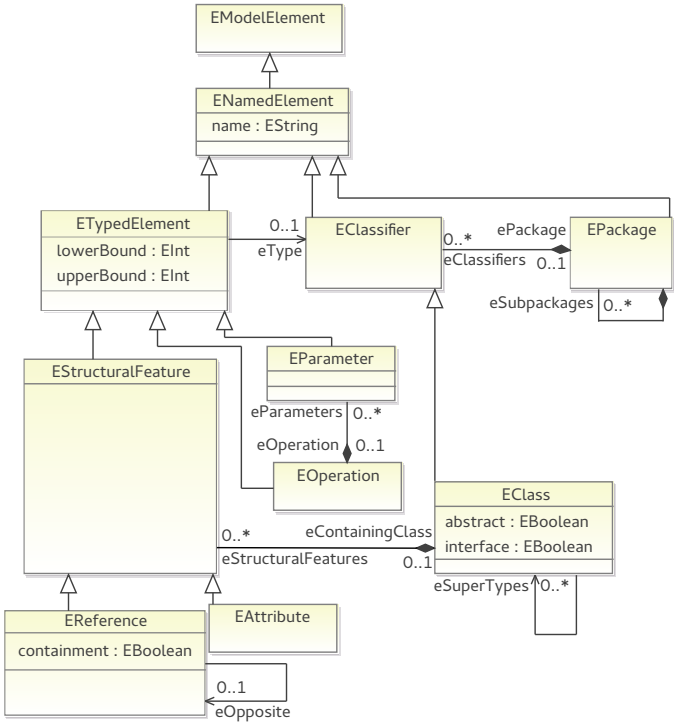

Figure 5: An excerpt of the Ecore metamodel used as the input metamodel for building the model slicer used within Explen

[3] Explen is freely available at the following address: https://github.com/arnobl/kompren/wiki.

```
1  slicer MetamodelSlicer {
2    domain: "platform:/plugin/org.eclipse.emf.ecore/
3             model/Ecore.genmodel"
4    input: ecore.EClass
5    radius: ecore.EClass
6    slicedClass: ecore.ENamedElement
7    slicedClass: ecore.EStructuralFeature feat
8      constraint: card1 [[ feat.lowerBound>0 ]]
9    slicedClass: ecore.EReference ref
10     constraint: composition [[ ref.containment ]]
11   slicedProperty: ecore.EClass.eSuperTypes option
12   slicedProperty: ecore.EClass.eSuperTypes option
13                   opposite(lowerTypes)
14   slicedProperty: ecore.EClass.eReferences option
15   slicedProperty: ecore.EClass.eAttributes option
16   slicedProperty: ecore.EClass.eOperations option
17   slicedProperty: ecore.ETypedElement.eType
18   slicedProperty: ecore.ENamedElement.name
19   slicedProperty: ecore.EReference.containment
20   slicedProperty: ecore.EClass.abstract
21   slicedProperty: ecore.EClass.interface
22   slicedProperty: ecore.ETypedElement.lowerBound option
23   slicedProperty: ecore.ETypedElement.upperBound option
24 }
```

Listing 1: The Kompren model slicer used in Explen

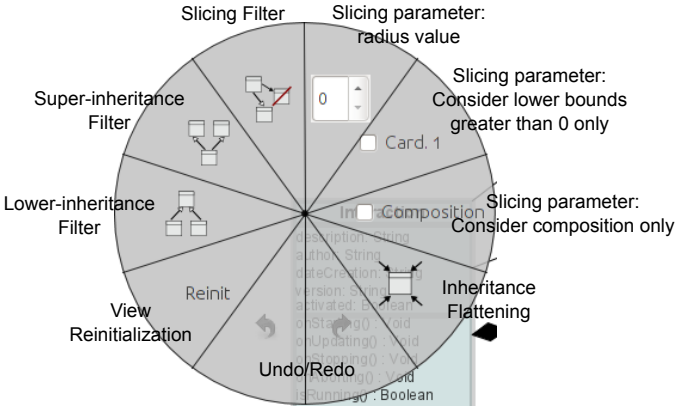

Figure 6: The Explen's context menu used to parameterize and launch the sliced-based visualization features

3.3.2. *Interactive Visualization Features for Metamodels*

The super and lower inheritance filters. These two filters show the super or lower inheritance tree of the targeted class. These two filters can be parameterized with one option, the radius effect (specified on line 5 in Listing 1). When activated with a given value greater than 0, this option shows the classes in relation with the targeted class by a distance equals or lower than the radius value. For instance, when the radius equals 1, only the direct lower or super classes of the targeted class are displayed. When set to 2, only these direct lower or super classes and their direct lower or super classes are shown. This option permits to reduce the number of classes shown in the view.

The keyword option (*e.g.* line 11) identifies the corresponding sliced element as optional. When compiled as a Java library, options become boolean parameters of the slicer. Developers can then programmatically call the slicer and state whether their corresponding class or property must be considered during the slicing. This feature permits to define multiple filtering features in a single model slicer. For instance, the super and lower inheritance filters do not consider the references defined between

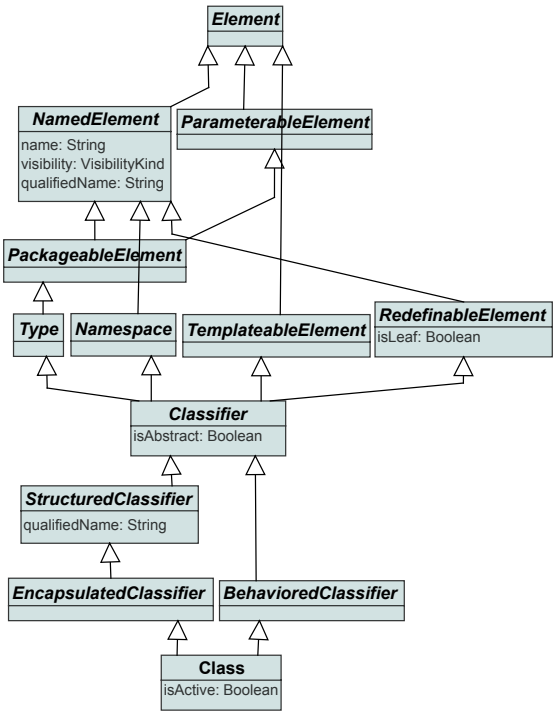

Figure 7: The super inheritance tree of the UML class *Class*

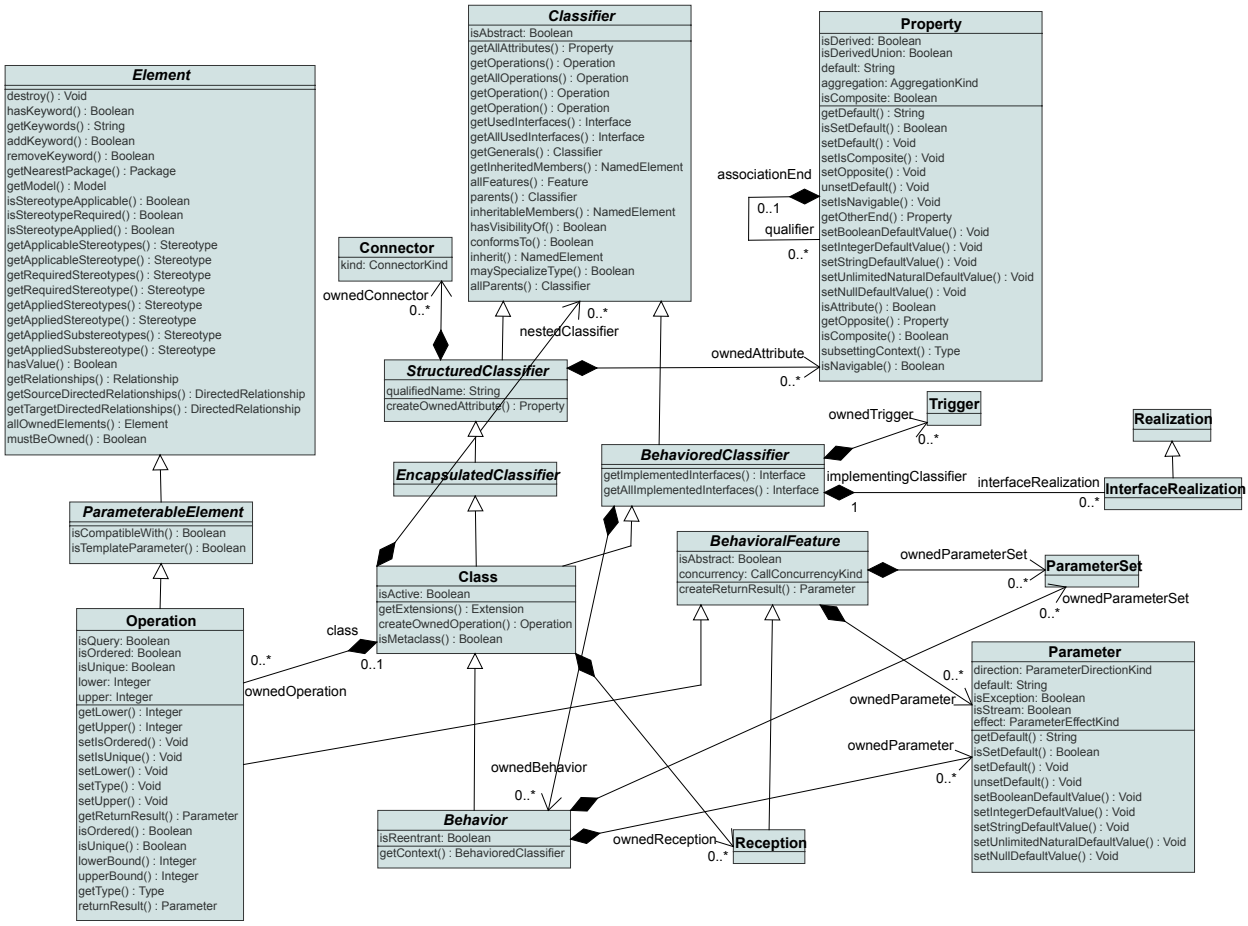

Figure 8: Slicing the UML metamodel using the class `Class` as input and parameterized with a radius of 3 and by slicing composition references only

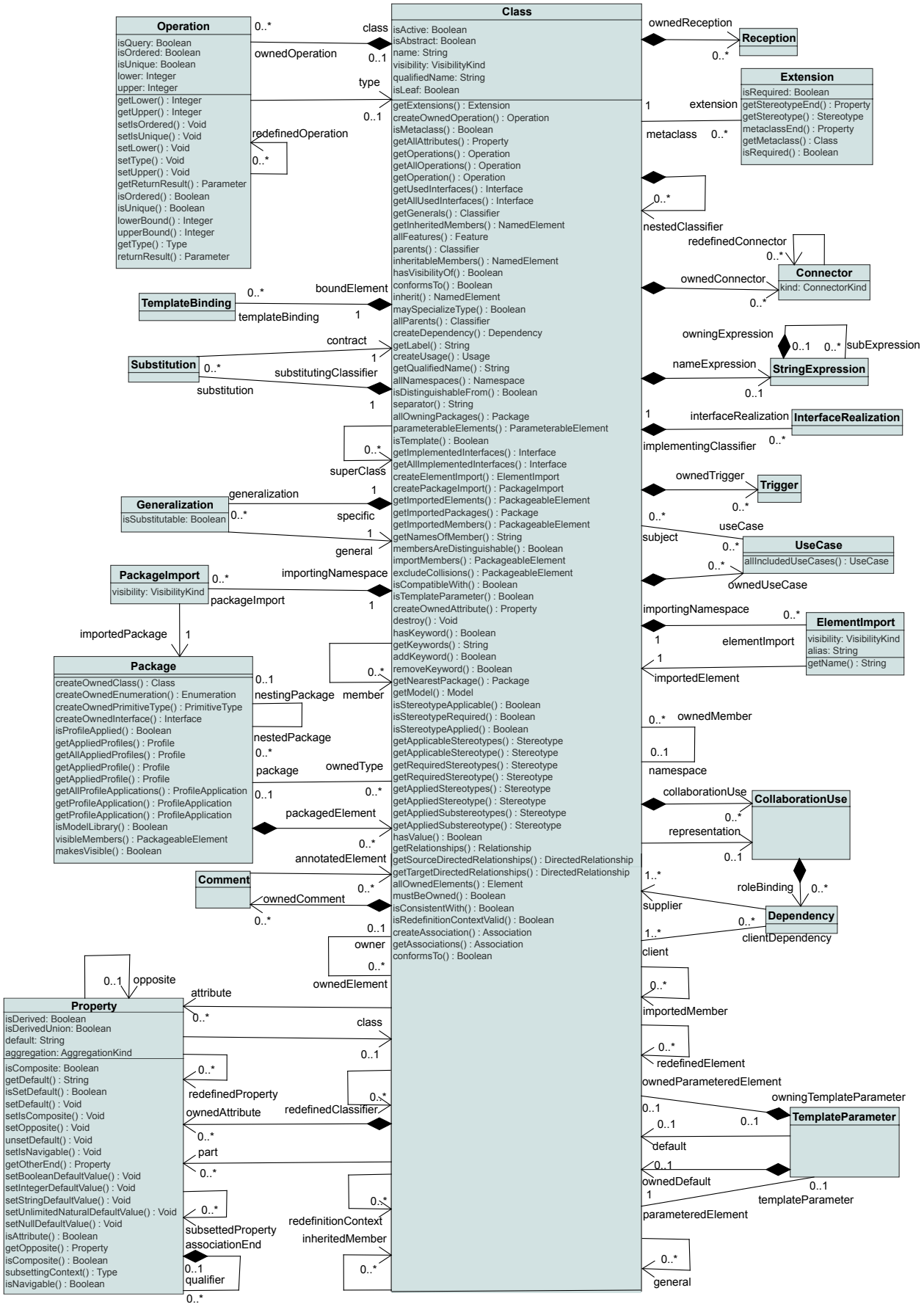

Figure 9: Flattening of the UML class *Class* followed by a slicing parameterized with a radius of 1

classes. So, when users click on the buttons dedicated to these filters (see Figure 6), several optional elements and their related elements of the slicer are not considered. The keyword `slicedProperty` selects the references or attributes to slice. The reference `eSuperTypes`, from the class `EClass`, is used to extract both the super and lower inheritance tree of a given class (lines 11 to 13). The keyword `opposite` (line 13) permits to navigate through a given reference in its opposite way when no opposite is defined (lower inheritance in our case). The keyword `slicedClass` selects the classes of the metamodel to slice. Here, the root class `ENamedElement` (line 6), and automatically all its lower classes, are sliced.

Modelers can use these filters to understand the inheritance hierarchy of a given class and identify the attributes and operations this class can access (Task 2, section 2). fig. 7 depicts such a result when focusing on the super inheritance tree of the class *Class*. Due to constraints on space, the classes' operations have been hidden, and the classes have been re-layouted manually to reduce the spacing.

The slicing filter. This filter hides classes not in relation (inheritance or references) with the targeted class. This filter has three options that can be combined. The first option slices composition references only. To do so, a constraint has been defined on the class to slice `EReference` (lines 9 and 10). Constraints are optional predicates that must be respected to trigger the slicing of the element targeted by the constraint. The constraint line 10 states that the boolean attribute `containment` of the class `EReference` must equals *true* to slice the class. The second option permits to slice references and attributes having their minimal cardinality greater than 0 only. A second constraint, line 8, permits to slice such references. The third option is the radius parameter (line 5).

Task 3 consists in showing the classes that are in relation with the UML class `Class` only. This task can be performed using the `Explen`'s slicer. To show only classes closely related to the class `Class`, the radius is set to 3. We also configure the slicer to consider composition references only. Then, the slicer is applied on the class `Class`, and the classes not sliced are hidden. Figure 8 shows the result of this slicing where only 18 classes among the 246 others are displayed. The resulting classes have been manually re-layouted.

The flattening filter. The super hierarchy of the targeted class pushed down into it: all its inherited attributes and relations now appear in the target class. To perform Task 2, the modeler can also flatten this hierarchy to put into *Class* all the properties and operations of its super-classes. Figure 9 shows the result of the flattening of *Class*. All the super-classes of *Class* have been removed while their properties and operations have been moved into *Class*. For instance, the goal of Task 1 is to show classes in direct relationship with *Class*. The modeler can accomplish this task with our viewer by restricting the radius effect of the slicer using the user interface: when the radius effect is set to 1, only classes in direct relationship with the sliced class are shown. Figure 9 also illustrates such successive combinations where the flattening filter is followed by a slicing of `Class` parameterized with a radius of 1.

Semantic zooming. The physical zoom is supplemented with a semantic zoom that shows different metamodel elements depending on the zoom level. When zooming out at 50 %, the attributes, operations, and roles are no more displayed (lines 15 to 16, 22 to 23 of Listing 1). The goal of this feature is to lighten or complete the amount of information shown to the user when visualizing a part of a metamodel at a given zoom level. The metamodel elements (operations, roles, cardinalities, and attributes) displayed at each zoom level has been defined empirically during the development phase of `Explen`.

All these interactive visualization features can be successively combined, undone, and redone, moving the current viewpoint on the canvas to its former position. `Explen` also provides a text field supplemented by auto-completion to search for a class. Selecting one class using this text field centers the viewpoint on the targeted class.

We developed the layout and the graphical library used by `Explen` to display metamodels (see for instance Figures 8 and 9) following the standard recommendations (abstract class name in italic, inheritance relations follow a bottom-up layout, *etc.*) also respected by mainstream metamodeling tools. As explained in the introduction, a study we conducted on 3462 well-formed Ecore metamodels highlighted that 82 % of these metamodels are composed of a single package, the mandatory root package. So, the current version of `Explen` does not display packages or provides interactive visualization features based on packages.

4. Experimental Design

This section presents the contribution related to the visualization techniques, provided by our editor `Explen`, to the understanding of metamodels by MDE stakeholders. Providing users with multiple kinds of views (*e.g.* 2D or tree views) may improve the exploration of metamodels. In this work we focus on the standard 2D representation of metamodels to study the impact of our proposed techniques. This contribution is compared to one of the most used metamodel editors, namely `EcoreTools` from EMF. The performance level is determined by the percentage of correct answers and the time and navigation effort spent by the subjects to perform the proposed tasks. A subject's effort is evaluated by capturing his/her interactions with the system, such as the number of scrolls or mouse moves.

`EcoreTools` has been selected after we compared the most widespread tools dedicated to the design of metamodels or domain models to identify their interactive navigation features. Based on these observations, summarized in Table 1, we selected the tool with the most interactive navigation features, namely `EcoreTools`. `EcoreTools`, `UML Designer`, and `Papyrus` are all based on Eclipse and share several similar features. `IBM Rational Rhapsody` can create a new diagram from one selected class. Such a new diagram displays all the classes and relations connected with the selected classes (super/lower inheritance, references). `Visual Studio` does not provide interactive visualization features but allows removing an element from the view without modifying the model.

Name	Version	Metamodel representation	Interactive visualization features	Address
EcoreTools	1.1	2D, tree	Show/hide elements, auto-layout, remove elements from the view, hide/show all relations	http://www.eclipse.org/ecoretools/
UML Designer	3.0	2D, tree	Show/hide elements manually, auto-layout, remove elements from the view	http://marketplace.obeonetwork.com/module/uml/download
Papyrus	1.0	2D, tree	Show/hide elements manually, auto-layout, remove elements from the view	http://www.eclipse.org/papyrus/
IBM Rational Rhapsody	8.1	2D, tree	Remove from view, select types of elements to show/hide, create new diagrams focusing on one class and its relations	http://www-03.ibm.com/software/products/en/ratirhaparchforsoft
Visual Studio	2012 Ultimate	2D, tree	Remove elements from the view	http://www.visualstudio.com/
MetaEdit+	5.0	forms	N/A	http://www.metacase.com/
MPS	3.1	text	N/A	http://www.jetbrains.com/mps/

Table 1: Tools dedicated to the design of metamodels or domain models and their interactive visualization features

MetaEdit+ and MPS do not provide modelers with 2D graphical editors for designing metamodels.

4.1. Objects

The objects of our experiments are two metamodels developed by third parties. We selected two metamodels to diversify the observations and thus limit the mono-method threat to validity. The first metamodel is the UML metamodel composed of 256 classes and 583 relations and attributes contained in a single package [1]. UML has been selected for its high number of elements. It is thus a relevant metamodel benchmark for evaluating visualization and navigation features. The second one is the RAM (*Reusable Aspect Models*) metamodel composed of 61 classes and 135 relations and attributes contained in a single package [27]. RAM has been selected for its moderate size, even though it is large enough not to be entirely visible using a standard screen. Both RAM and UML concern software engineering but focus on different concerns.

4.2. Hypotheses

The main questions about the efficiency of our slice-based visualization features are described as follows:

Q1 Do these visualization features reduce the time needed to complete typical tasks?

Q2 Do the visualization features, provided by Explen, improve the correctness of those tasks performed on metamodels?

Q3 Do these visualization features reduce the navigation effort needed to complete those tasks?

From these three questions can be inferred the following null hypotheses:

H_{0_1} There is no difference between the subjects using EcoreTools and subjects using Explen in the average time they needed to complete the provided tasks.

H_{0_2} There is no difference between the subjects using EcoreTools and the subjects using Explen in the average correctness of their answers given to complete the provided tasks.

H_{0_3} There is no difference between the subjects using EcoreTools and subjects using Explen in the average navigation effort spent to complete the provided tasks.

If these null hypotheses are rejected, alternative hypotheses are defined as follows:

H_{1_1} *The average time needed to complete tasks on metamodels is lower for the subjects using Explen than for the subjects using EcoreTools.* Explen is built on top of visualization features that make it possible to focus on elements relevant to the ones selected by the users. That may help users to achieve their goals quicker than using EcoreTools.

H_{1_2} *The average correctness of the answers given to complete tasks on metamodels is better for the subjects using Explen than for the subjects using EcoreTools.* This first alternative hypothesis is motivated by the fact that the Explen's visualization features aim at reducing the large amount of information provided to users by showing elements relevant to the current situation only. As a result, that may lead to more accurate answers.

H_{1_3} *The average navigation effort needed to complete tasks on metamodels is lower for the subjects using Explen than for the subjects using EcoreTools.* The rationale behind this last alternative hypothesis is that compared to EcoreTools, Explen provides users with dedicated interactive navigation features that aim at easing the navigation through metamodels.

4.3. Dependent Variables

In addition to the time to perform tasks and their correctness, we also collected objective variables that capture the navigation effort.

- *Average Time* (TIME): measures the average time in seconds the subjects spent to give their answers for each task.
- *Correct Answer* (CORR): measures the correctness of each task answered by a subject.
- *Average Mouse Move* (MOVE): measures the average number of mouse moves a subject did per tasks. This variable will be used to measure the navigation effort.
- *Average Scrolling Actions* (SCROLL): measures the average number of scrolling actions (using the mouse scrolls or the scroll bars) a subject did per tasks. This variable will be used to measure the navigation effort.

All the answers are composed of a set of names (class, relation, or attribute name). The correctness of an answer is computed using the following formula:

$$\frac{|correct\ names|}{|expected\ names| + |incorrect\ names|} * 100 \quad (1)$$

where incorrect names provided by subjects reduce the correctness of the answer.

The size and the resolution of the screen, and the input device used by each subject for the experiments are controlled variables since we provide subjects with the same computer, system, and monitor (24-inch, 1920×1200 resolution) with a mouse. The subjects could choose either an AZERTY or a QUERTY keyboard before the experiments to limit typing mistakes.

4.4. Data Collection

The evaluation study takes the form of a comparison between two tools[4]. The first tool is `EcoreTools` (version 1.1.0.201205150811). `EcoreTools` runs on top of Eclipse as a plug-in. It offers a 2D view of the edited metamodel (see Figure 10). It provides users with standard graphical editing features such as a physical zoom and the possibility to move elements using a pointing device. `EcoreTools` provides two filtering features accessible from the context menu. These filters hide the inheritance or the reference relations of the whole metamodel.

The second tool is `Explen` (version 1.0) we developed in the purpose of this research work. `Explen` is introduced in Section 3.

For the purpose of the experiments, both `EcoreTools` and `Explen` have been supplemented with the same panel on the right of their user interface (see Figures 10 and 11). This panel displays the current question asked to the subject and a text field to give an answer. When a new question is asked, the metamodel and the navigation features are not available and visible, and no information is recorded. Once ready, the subject clicks on the button "*Start*" that shows the metamodel, activates the navigation features, triggers the information recorder, and shows a text field to answer the current question. The subject can then navigate into the metamodel to answer the current question. This process permits the subject to focus on the question before answering it and to ask the experimenter for clarification if needed. The subject can then give her answer to the current question using the dedicated text field. The subject can correct her answer until she clicks on the button "*Validate Answer*". Clicking on this button results in saving the answer and the information recorded during the time-slot spent to give the answer. Then, the answer text field, the button "*Validate Answer*", and the metamodel are hidden. Moreover, this action disables the navigation features and shows the next question with the button "*Start*". This process is repeated until all the questions are answered. The metamodel view is re-initialized between each question.

During each time-slot framed by the click on the buttons "*Start*" and "*Validate Answer*", the subjects' activity is recorded. The subjects have been notified about the recording before the experiments. The recorded information are composed of the number of time that: the mouse is moved; the scroll bars are moved; the mouse scroll is used; the `Explen`'s visualization features are used. The time between the click on the two buttons is also measured.

Several questions are asked at the end of the experiments to the subjects through a questionnaire integrated in the GUI of the latest used editor. These questions relate to the age, the current position (Master student, PhD student, engineer, or assistant professor), and the experience in MDE, `EcoreTools`, UML, and RAM.

4.5. Tasks

Four questions (tasks) are successively asked to the subjects through the panel integrated in the two metamodel viewers. These tasks concern different parts of the studied metamodels. They have been defined to evaluate various kinds of comprehension tasks.

4.5.1. Task Design

The tasks have been defined from a survey we conducted on subjects working in the MDE community. This study aims at identifying the main tasks that MDE stakeholders perform while handling metamodels. This survey was conducted in two steps. The goal of the first one is to identify tasks that MDE stakeholders perform when handling metamodels. To this end, we designed a first web questionnaire that consists in evaluating the importance of 14 metamodeling tasks ("important" *vs* "not important"). To be more complete, we encouraged the subjects to suggest other non-listed tasks. The tasks that received at least 70 % of "important" answers were retained. The tasks frequently suggested by different subjects have also been added. This web questionnaire was filled by 68 subjects. These subjects were composed of researchers (74 %), PhD students (12 %), industrials (10 %), research engineers (3 %), and master students (1 %). They claimed to be expert (37 %), proficient (37 %), competent (15 %), advanced beginner (7 %), and novice (4 %) in MDE. Out of the 14 proposed tasks, 8 have been retained in addition to 2 tasks suggested by the subjects.

[4]All the material of the experiments is freely available on the following web page: `https://github.com/arnobl/kompren/wiki/Data-of-the-Experiment-on-Explen`

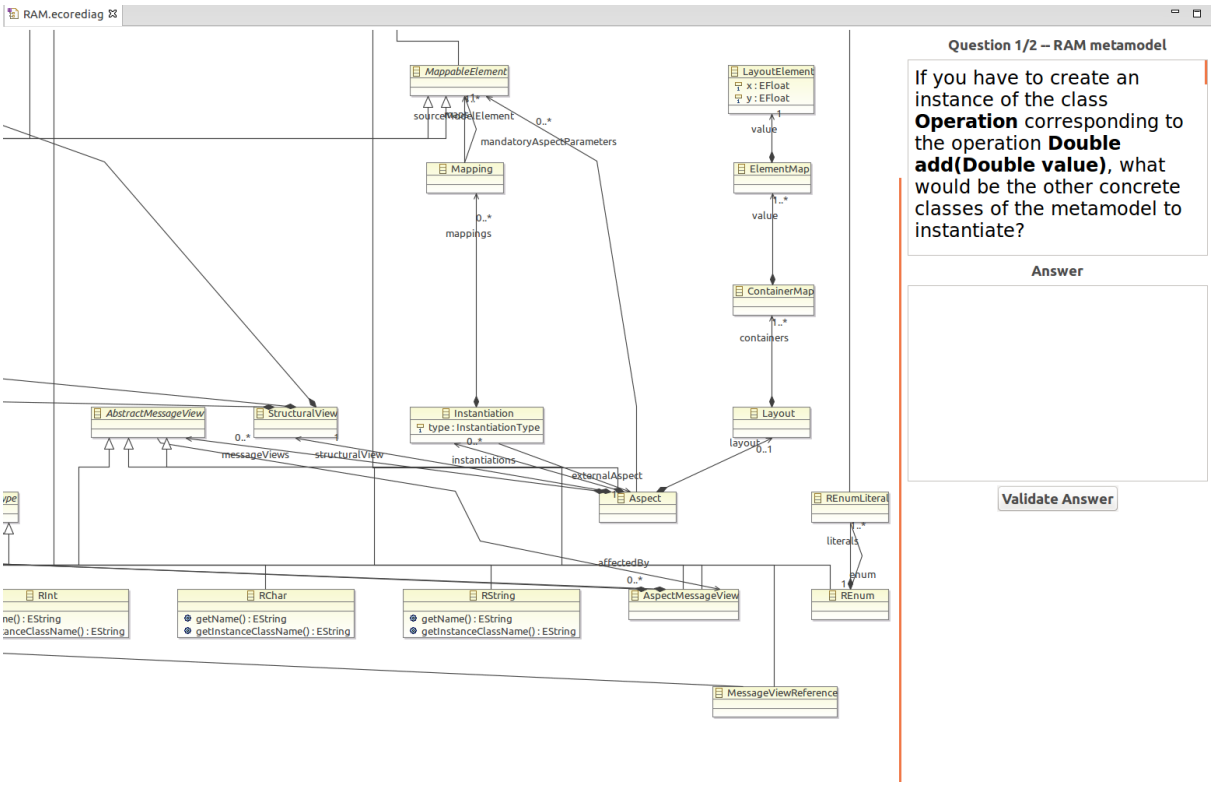

Figure 10: `EcoreTools` supplemented with a panel (on the right) for the experiments

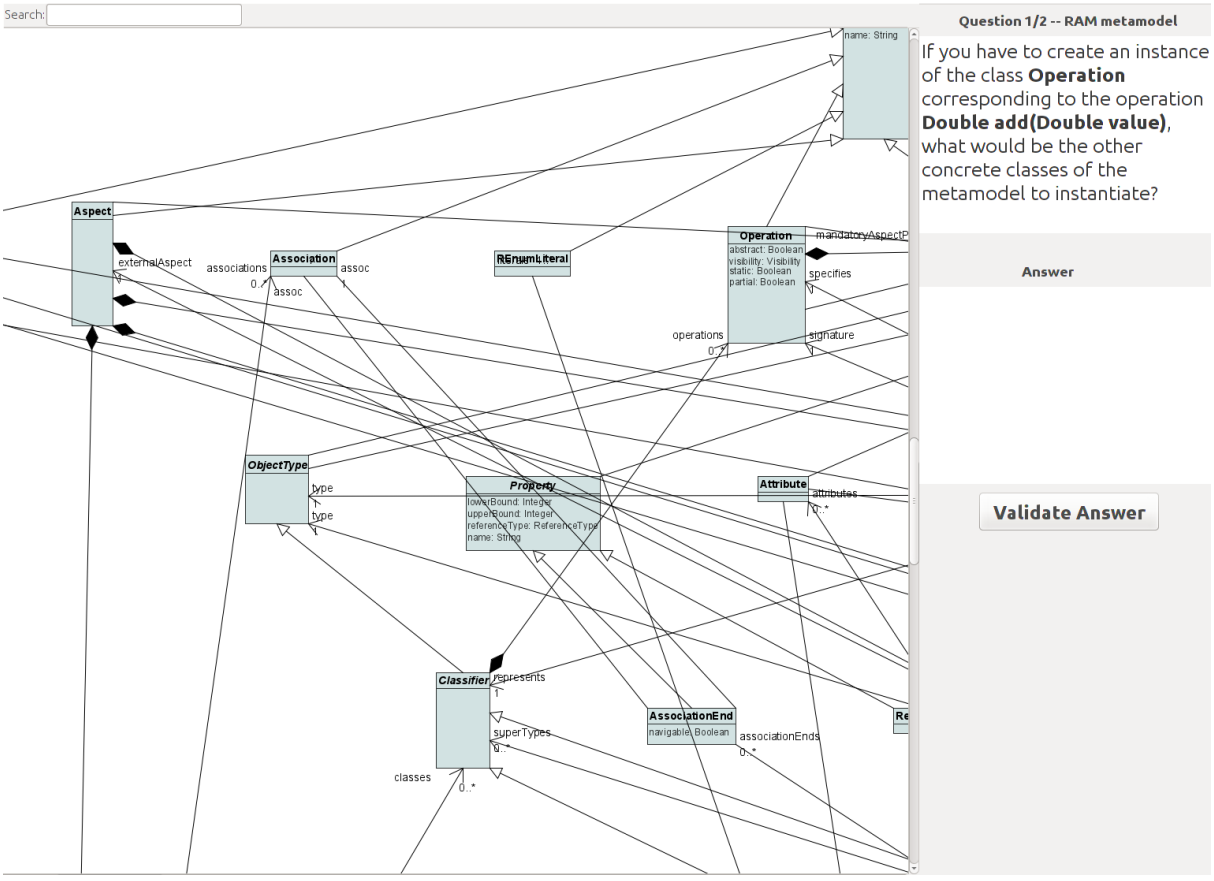

Figure 11: `Explen` supplemented with a panel (on the right) for the experiments

A second web questionnaire was designed to order, by importance, the 10 tasks resulting from the first questionnaire. The subjects had to rate the importance of each task on a 1-to-5 scale where 5 stands for "very relevant". The second questionnaire was filled by 60 subjects. These subjects were composed of researchers (66 %), PhD students (14 %), industrials (11 %), research engineers (4 %), and others (5 %). They claimed to be expert (41 %), proficient (39 %), competent (13 %), advanced beginner (4 %), and novice (4 %) in MDE.

Ranking	Task	Average importance
1	Defining a concrete syntax for a metamodel	4.32
2	Instantiating a metamodel	4.30
3	Transforming models into text or code	4.25
4	Maintaining a metamodel	4.20
5	Creating editing tools for a metamodel	4.11
6	Transforming elements of a metamodel into elements of another one	4.02
7	Defining invariants and/or constraints on metamodel importance	3.86
8	Identifying how elements of a metamodel are linked to others (of the same metamodel)	3.80
9	Analyzing that a metamodel is well designed	3.57
10	Comparing metamodels	3.10

Table 2: The ranking of the selected tasks that MDE stakeholders perform while handling metamodels

The average importance of the 10 tasks is given in Table 2. From this table, we make the following observations that led to the identification of four categories of tasks. First, all these tasks require to *learn or understand* the metamodel under study. This understanding can be partial (*e.g.* for developing a model transformation) or complete (*e.g.* for developing a concrete syntax). Second, *refactoring* takes an important role in these tasks: maintaining a metamodel implies maintaining all its associated tools and model transformations. Third, metamodel *instantiation* is an important activity underlying most of these tasks. For instance, metamodel editors provide facilities for instantiating metamodels. Last, metamodel *quality* raises interest from the subjects.

4.5.2. Tasks Description

Our experiments consist of subjects using two different metamodel visualization tools to observe whether their understanding of metamodels is improved when interactive visualization techniques are provided. Therefore, our experiments must rely on visualization and navigation tasks for identifying the metamodel elements (classes, roles, *etc.*) required to perform one of the 10 identified tasks. Following the 10 tasks and the four categories we identified, we designed four tasks. Each of these four tasks has one variant. All these tasks have been adapted to the metamodel under study (UML and RAM). It implies that if the question remains unchanged, the metamodel elements to identify may vary. These tasks are defined as follows:

Tasks related to *metamodel instantiation*.

T_{1a} "*If you have to create an instance of the class* Operation *corresponding to the operation* `Double add(Double value)`, *what would be the other concrete classes of the metamodel to instantiate?* This task is asked identically for both the UML and RAM metamodels since the concept of operation is shared by these two metamodels but with different designs. The purpose of this task is to evaluate how subjects can identify concepts in a metamodel used by a specific model example.

T_{1b} *"If you have to create an instance of the class* ConditionalNode*, what would be the mandatory concrete classes in relation with* ConditionalNode *that must be also instantiated?"*
This question is for UML. For RAM, the class of interest is *Message*. The purpose of this task is to evaluate the ability of the subjects to explore the metamodels to identify the required metamodel elements. Mandatory classes are either classes directly or indirectly in relation with *ConditionalNode* with a min cardinality greater than 0, or concrete sub-classes of mandatory abstract classes. This definition is explained during the tutorial starting each experiment and is available all along it.

Tasks related to *metamodel refactoring*.

T_{2a} "*List the name of the abstract classes in the super class hierarchy of* FlowFinalNode *that are not doing much (*i.e. *that do not contain attributes, operations, and output references/compositions) and that can be removed*".
This question is for UML. For RAM, the class of interest is *Class*. The purpose of this task to evaluate the ability of the subjects to identify a bad smell, *Speculative Generality* [28] in this case.

T_{2b} *"Give the name of the redundant attribute (direct or inherited) of the class* Device *(*i.e. *same name and type)"*
This question is for UML. For RAM, the class of interest is *Reference*. The purpose of this task to evaluate the ability of the subjects to identify a bad smell, *Duplicated Code* [28] in this case.

Tasks related to *metamodel quality*.

T_{3a} "*The classes* Actor *and* Trigger *are coupled only by one unique reference via another class. Give the name of this reference that would make these classes independent if removed.*"
This question is for UML. For RAM, the classes of interest are respectively *Aspect* and *Type*. The purpose of this task is to evaluate the ability of the subjects to navigate through the relations and classes linked to a given class in order to perform metamodel changes.

T_{3b} *Give the name of at least one class that has a high number of incoming and a high number of outgoing references compared to the other classes*." The purpose of this task is to evaluate the ability of the subjects to identify bottleneck classes of a metamodel.

Tasks related to *metamodel understanding*.

T_{4a} "*Give the name of at least one intermediate class between the class* State *to the class* Transition."
This question is for UML. For RAM, the classes are *LifeLine* and *FragmentContainer* respectively. The purpose of this task is evaluate the ability of subjects to navigate from one class to another one using relations.

T_{4b} *"Enumerate the name of all the attributes (direct or inherited) of the class* Feature*."*
This question is for UML. For RAM, the class is *RInt*. The purpose of this task is to evaluate the ability of the subjects to navigate through the inheritance tree of a given class.

4.6. Procedure

Before the experiments, the procedure and the tools have been tested on several subjects. Several bugs in the question panels (that supplemented each tool for the experiments) have been identified and corrected and several task descriptions have been precised. These tests also permitted to have an estimation of the time required to performed each task. Then, the experiments have been performed following the procedure described in this section.

Tools	`EcoreTools`		`Explen`	
Metamodels	UML	RAM	UML	RAM
Group 1	T_{1a}	T_{2a}	T_{3a}	T_{4a}
Group 2	T_{2b}	T_{3a}	T_{4b}	T_{1a}
Group 3	T_{3b}	T_{4b}	T_{1b}	T_{2b}
Group 4	T_{4a}	T_{1b}	T_{2a}	T_{3b}

Table 3: Distribution of the tasks.

The experiments were conducted over three days during several sessions on 32 subjects. Each session involved between two and four subjects. We requested subjects not to talk about the experiments until its end. Each subject performed four successive tasks. This number of tasks was defined to limit the duration of the experiments on one subject to around 20 minutes. As depicted by Table 3, the subjects were clustered in four groups (eight subjects per group) to vary the execution order of the tasks on each metamodel and each tool. So, each group performed the four tasks but not using the same metamodel and tool for each task. For each group, two tasks (one with UML and another one with RAM) have been performed using `EcoreTools`, and similarly two others using `Explen`. Each of the eight tasks were executed two times by different groups on different metamodels and tools. The execution order, depicted by Table 4, of the four tasks has been manually randomized for each subject of a group.

Each session started with an explanation about the experiment and what the subjects had to do. The subjects were also notified that: anonymized data would be recorded; there is no time limit to perform the tasks but around 20 minutes should be enough; there is no reward. A 2-page document was provided to the subjects. It describes the user interface of the two tools, their features, and how the `Explen`'s features work. Before the experiments, the subjects could train during around 10 minutes on the two tools on metamodels and questions that differ from those of the experiments. The subjects that already know `EcoreTools` could spend their training time on `Explen` to balance the proficiency on the two tools. Once the experiments finished, a form appeared in the user interface of each tool to gather information about the subjects (see Section 4.4). Finally, we proposed to each subject to write informal and anonymous feedback about the tools and the experiments.

Subjects	Group 1				Group 2				Group 3				Group 4			
S_{i1}	T_{1a}	T_{2a}	T_{3a}	T_{4a}	T_{2b}	T_{3a}	T_{4b}	T_{1a}	T_{3b}	T_{4b}	T_{1b}	T_{2b}	T_{4a}	T_{1b}	T_{2a}	T_{3b}
S_{i2}	T_{1a}	T_{2a}	T_{4a}	T_{3a}	T_{2b}	T_{3a}	T_{1a}	T_{4b}	T_{3b}	T_{4b}	T_{2b}	T_{1b}	T_{4a}	T_{1b}	T_{3b}	T_{2a}
S_{i3}	T_{2a}	T_{1a}	T_{3a}	T_{4a}	T_{3a}	T_{2b}	T_{4b}	T_{1a}	T_{4b}	T_{3b}	T_{1b}	T_{2b}	T_{1b}	T_{4a}	T_{2a}	T_{3b}
S_{i4}	T_{2a}	T_{1a}	T_{4a}	T_{3a}	T_{3a}	T_{2b}	T_{1a}	T_{4b}	T_{4b}	T_{3b}	T_{2b}	T_{1b}	T_{1b}	T_{4a}	T_{3b}	T_{2a}
S_{i5}	T_{3a}	T_{4a}	T_{1a}	T_{2a}	T_{4b}	T_{1a}	T_{2b}	T_{3a}	T_{1b}	T_{2b}	T_{3b}	T_{4b}	T_{2a}	T_{3b}	T_{4a}	T_{1b}
S_{i6}	T_{3a}	T_{4a}	T_{2a}	T_{1a}	T_{4b}	T_{1a}	T_{3a}	T_{2b}	T_{1b}	T_{2b}	T_{4b}	T_{3b}	T_{2a}	T_{3b}	T_{1b}	T_{4a}
S_{i7}	T_{4a}	T_{3a}	T_{1a}	T_{2a}	T_{1a}	T_{4b}	T_{2b}	T_{3a}	T_{2b}	T_{1b}	T_{3b}	T_{4b}	T_{3b}	T_{2a}	T_{4a}	T_{1b}
S_{i8}	T_{3a}	T_{4a}	T_{1a}	T_{2a}	T_{1a}	T_{4b}	T_{3a}	T_{2b}	T_{2b}	T_{1b}	T_{4b}	T_{3b}	T_{3b}	T_{2a}	T_{1b}	T_{4a}

Table 4: Randomization of the execution order of the tasks for each subject of each group. For technical reasons, the tasks performed using the same tool must be grouped.

5. Analysis and Results

The results of the experiments are analyzed and discussed in this section. This analysis focuses on the three research questions introduced in Section 4.2:

Q1 Do these visualization features reduce the time needed to complete those tasks?

Q2 Do the visualization features, provided by `Explen`, improve the correctness of typical tasks performed on metamodels?

Q3 Do these visualization features reduce the navigation effort needed to complete those tasks?

Dependent variable	**Mean (#) `Explen`**
$PRUNING_{T1}$	4.1
$PRUNING_{T2}$	3.8
$PRUNING_{T3}$	4.3
$PRUNING_{T4}$	1.3
$PRUNING_{All}$	3.3
$HIERARCHY_{T1}$	1.2
$HIERARCHY_{T2}$	1.6
$HIERARCHY_{T3}$	0.4
$HIERARCHY_{T4}$	0.4
$HIERARCHY_{All}$	0.9
$FLAT_{T1}$	0.8
$FLAT_{T2}$	0.9
$FLAT_{T3}$	0.4
$FLAT_{T4}$	3.9
$FLAT_{All}$	0.6

Table 5: `Explen` interactive features usage.

Dependent variable	Mean (#) `Explen`	Mean (#) `EcoreTools`	Normal Distrib. ?	Mean (#) Diff	Significance p-value
$MOVE_{T1}$	5735	5909	N	-173	.792
$MOVE_{T2}$	3894	4443	N	-549	.113
$MOVE_{T3}$	4138	3772	N	366	.763
$MOVE_{T4}$	1152	4505	N	-3353	<.001
$H_{0_3}: MOVE_{All}$	3730	4657	N	-927	0.1
$SCROLL_{T1}$	212	962	N	-750	.002
$SCROLL_{T2}$	177	716	N	-538	.004
$SCROLL_{T3}$	137	394	N	-257	.003
$SCROLL_{T4}$	7	588	N	-581	<.001
$H_{0_3}: SCROLL_{All}$	133	665	N	-531	<.001

Table 6: Mouse Usage.

We apply the independent samples t-test and Mann-Whitney tests [29] to compare the performance, in terms of time and correctness, and the navigation effort of the two tools, using a 95 % confidence level (*i.e.* p-value$<$0.05). These both statistical tests are based on the null hypothesis and assess whether two independent populations are the same against an alternative hypothesis. In particular, they assess that one of the populations tends to have larger average values than the other one. The Mann-Whitney test makes no assumptions about the distributions of assessed variables whereas the independent samples t-test applies on normal distributions.

The discussion of the results is based on data obtained from the measured dependent variables (Tables 5 and 6).

5.1. Time Analysis

Dependent variable	Mean (s) `Explen`	Mean (s) `EcoreTools`	Normal Distrib. ?	Mean (s) Diff	Significance p-value
$TIME_{T1}$	311	375	Y	-64	0.31
$TIME_{T2}$	200	259	Y	-59	0.44
$TIME_{T3}$	213	214	Y	-1	0.99
$TIME_{T4}$	73	288	Y	-215	<0.001
$H_{0_1}: TIME_{All}$	199	284	N	-85	0.04

Table 7: The average time measured for each task.

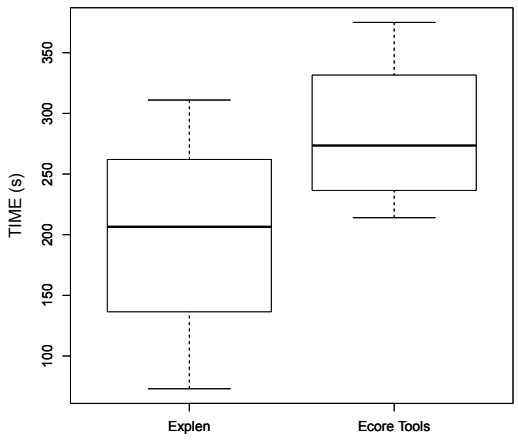

Figure 12: Comparison between `Explen` and `EcoreTools` in terms of time.

Table 7 summarizes the average time measured for each task during the experiments. Figure 12 provides a visual representation of these results in the form of boxplots. Regarding Task T_1, the average time measured is in favor of `Explen` (-64 s less) but it is not statistically significant. T_1, which deals about metamodel instantiation, requires to explore metamodels to identify the required metamodel elements.

Task T_2 exhibits similar results than T_1. T_2, which deals about metamodel refactoring, corresponds to the identification of bad smells. It requires to explore the super inheritance tree to find duplicated attributes and useless abstract classes.

Task T_3, which deals about metamodel quality, does not highlight any statistically significant difference in the average time (1 second) between `Explen` and `EcoreTools`. One possible explanation is that the interactive features we propose may not be adequate for this task.

The average measured time of Task T_4 is clearly in favor of `Explen` (more than 3 minutes less) and statistically significant. T_4, dedicated to metamodel comprehension, consists of identifying metamodel elements in relation with a given class.

When considering all the tasks, the average time is in favor of `Explen` and statistically significant with a p-value of 0.04 (< 0.05). This can be explained by the reduced mouse usage in favor of the `Explen`'s interactive features. Therefore, we can reject the null hypothesis H_{0_1} "*There is no difference between the subjects using `EcoreTools` and subjects using `Explen` in the average time they needed to complete the provided tasks*" and accept the alternative hypothesis H_{1_1} "*The average time needed to complete tasks on metamodels is lower for the subjects using `Explen` than for the subjects using `EcoreTools`*".

5.2. Correctness Analysis

Dependent variable	Mean (%) `Explen`	Mean (%) `EcoreTools`	Normal Distrib. ?	Mean (%) Diff	Significance p-value
$CORR_{T1}$	25	23	Y	2 %	0.85
$CORR_{T2}$	71	22	N	49 %	0.01
$CORR_{T3}$	56	70	N	-14 %	0.51
$CORR_{T4}$	99	48	N	51 %	0.02
$H_{0_1}: CORR_{All}$	63	41	N	22 %	0.03

Table 8: The average correctness of each task.

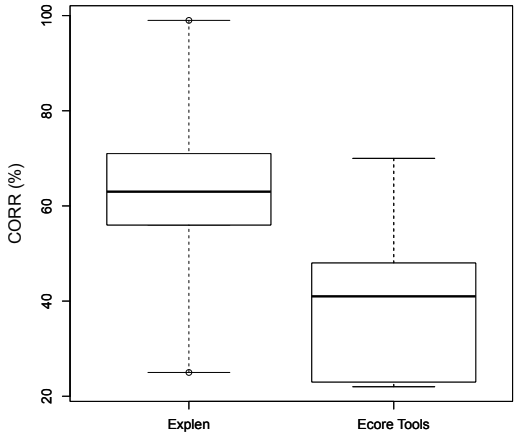

Figure 13: Comparison between `Explen` and `EcoreTools` in terms of correctness.

Regarding the correctness of the tasks performed by the subjects (Table 8 and Figure 13), the results follow the trend of the time analysis with one relative difference concerning Task T_1 related to metamodel instantiation. The correctness of this task does not show a significant difference between `Explen` and `EcoreTools`. The correctness for this task is rather low for both these tools (resp. 25 % and 23 %). One explanation of these low results might be the difficulty to execute Task T_1, which requires more thought and exploration of the metamodels, compared to the other tasks, which are more specific and straightforward to execute.

Task T_2 for metamodel refactoring exhibits better correctness results for `Explen` (71 % *vs* 22 %, statistically significant). T_2 is the task where subjects used in average the most `Explen` interactive features (3.8 prunings, 1.6 hierarchies, 0.9 flats). These features allow refining the search and perform with accuracy the sub-tasks targeted.

Task T_3, which deals about metamodel quality, shows worst correctness results for `Explen` but this result is not statistically significant. T_3 consists of finding one class that has a high number of incoming and outgoing references, *i.e.* references that link two given classes. As the time analysis, an explanation may be the inadequacy of the interactive features of `Explen`.

Similarly to T_2, Task T_4 shows better correctness results for `Explen` (99 % *vs* 48 %) and is statistically significant. As mentioned in the time analysis, the use of the flattening of a class hierarchy may explain the advantage of `Explen`.

When considering all the tasks, correctness is in favor of `Explen` (63 % *vs* 41 %) and is statistically significant. Therefore, we can reject the null hypothesis H_{0_2} "*There is no difference between the subjects using `EcoreTools` and the subjects using `Explen` in the average correctness of their answers given to complete the provided tasks*" and accept the alternative hypothesis H_{1_2} "*The average correctness of the answers given to complete tasks on metamodels is better for the subjects using `Explen` than for the subjects using `EcoreTools`*".

5.3. Navigation Effort Analysis

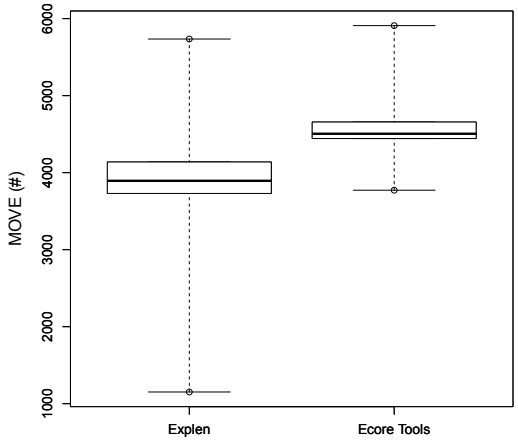

Figure 14: Comparison between `Explen` and `EcoreTools` in terms of mouse move.

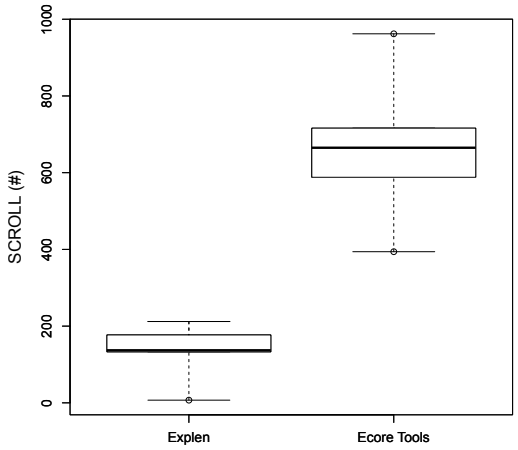

Figure 15: Comparison between `Explen` and `EcoreTools` in terms of mouse scroll.

Table 6 summarizes the average mouse move and mouse scroll measured for each task during the experiments. Figures 14 and 15 provide a visual representation of these results in the form of boxplots. All the results that concern the mouse scroll usage are statistically significant. Only T_4 and the average results of the mouse move usage are statistically significant.

Regarding T_1, using `EcoreTools` the subjects intensively used the scrolling features (Table 6: 962 scrolls). Using `Explen` this use decreased to 212. The mouse move results, however, are slightly in favor of `Explen` (5735 *vs* 5909). This may explain the different execution times as detailed in the previous sections. Moreover, we can observe that the use of the slicing features (4.1 pruning, 1.2 hierarchy, 0.8 flattening) may limit the use of the scrolling features.

Similarly to T_1, the subjects intensively used the scrolling features with `EcoreTools` on T_2 (Table 6: 716 scrolls) by opposition to `Explen` (177 scrolls). The number of mouse moves is also reduced using `Explen` (3894 *vs* 4443). The subjects produced less mouse moves and scrolls in T_2 compared to T_1. This may be explained by the fact that the common representation of super inheritance has to follow a bottom-up representation, which limits the search directions.

On T_3, the results regarding the mouse move usage is in favor of `EcoreTools` (3772 *vs* 4138). However, the mouse scroll usage is in factor of `Explen` (137 *vs* 394). It may mean that without a dedicated interactive navigation feature, the subjects explored the metamodels by moving the mouse.

Similarly to T_1 and T_2, on T_4 the subjects intensively used the scrolling features (588) and moved their mouse (4505) when using `EcoreTools`. Using `Explen`, the use of the scrolling features is reduced to 7 and the mouse moves to 1152. The reason is the use of the flattening feature (3.9) provided by `Explen` that reduces the need of scrolling by flattening a class hierarchy.

When considering all the tasks, mouse move and scroll usages are in favor of `Explen` (respectively 3730 *vs* 4657 and 133 *vs* 665) and are statistically significant. Therefore, we can reject the null hypothesis H_{0_3} "*There is no difference between the subjects using `EcoreTools` and subjects using `Explen` in the average navigation effort spent to complete the provided*

tasks" and accept the alternative hypothesis H_{1_3} "*The average navigation effort needed to complete tasks on metamodels is lower for the subjects using* `Explen` *than for the subjects using* `EcoreTools`".

5.4. Discussion

When considering all the tasks in terms of time, correctness, and navigation effort, they are in favor of `Explen` and are statistically significant. Depending on the task and when using `Explen`, the subjects used different `Explen` interactive features. The use of these interactive features may be the reason of the decreasing use of classical mouse interactions (moves, scrolls). Reduction of the number of visible classes and flattening of hierarchical trees are `Explen` features, which aim at limiting the navigation effort users have to do to explore parts of metamodels. The answers of the questionnaire provided to the subjects after the experiments confirm the benefits of the `Explen` features. The subjective comments of the subjects are that, in general, they appreciated the developed interactive features. For instance, some of the comments include: "*`Explen` is really helpful for software modelers to understand the relations among classes*", "*`Explen` helps users to navigate in the diagram*", "*using `EcoreTools` I gave up*". Some drawbacks were also mentioned: "*`Explen` did not help me when searching for high numbers of incoming/outcoming references*", "*Using the flattening feature large classes are still difficult to visualize*". The former follows our analysis of the results of T_3, while the latter correctly summarizes one limit of the flattening feature. Suggestions, related to the tasks, were also provided: *A feature for identifying the paths between two classes is missing*. This remark concerns Task T_3 were the subjects had to identify a reference between two classes.

Several subjects' comments suggested that complementing the 2D representation with other ones may be useful: "*Sometimes I use the `EcoreTools` tree view to navigate*". We agree that providing users with multiple kinds of views may improve the exploration of metamodels. In this work we focus on the standard 2D representation of metamodels to study the impact of our proposed techniques. We thus forbid the subjects to use the tree view representation that `EcoreTools` provides.

In summary, if the conducted experiments exhibit results in favor of `Explen`, this last has some limitations and requires other interactive visualization techniques. For instance, Task T_3 highlighted the need for interactive visualization techniques showing the common elements shared by several classes. However, `Explen` provides efficient capabilities when dealing with a subset of a metamodel inferred by inheritance and references relationships. It also provides attractive features for navigating through these two kinds of relationships.

5.5. Threats to Validity

Internal validity. The obtained results depend on the layout of the tools. Indeed, the algorithm layout and the drawing of the relations differ from `Explen` to `EcoreTools` (straight lines with `Explen`, multi-lines using `EcoreTools`). Moreover, `EcoreTools` is embedded with the Eclipse environment while `Explen` is a Java Swing application. However, we lessen this threat by closing various Eclipse panels and toolbars to present `EcoreTools` and `Explen` as similar as possible. In particular, we set the same dimensions of the diagram area for both tools. Moreover, the layout algorithms of `Explen` and `EcoreTools` follow the standard representation of metamodel (bottom-up inheritance, same symbols, *etc.*). The results also depend on the tasks performed during the experiments. Thus, we performed a survey among the MDE community to identify the most common and representative set of tasks performed by metamodelers so as not to influence the results.

External validity. This threat concerns the possibility to generalize our findings. We designed the experiments using two metamodels (UML and RAM) and 8 tasks to diversify the observations. The two metamodels selected, UML and RAM, have various sizes (respectively, 256 classes/583 relations and 61 classes/135 relations) and focus on different concerns. Moreover, the UML metamodel is widely used. The eight tasks, selected independently following a survey, relate also to different metamodeling concerns : instantiation, refactoring, quality, and understanding. Despite the efforts made to select multiple representative modeling tasks in our experiments, we do not pretend that for any task taken in isolation `Kompren` will give better results that `EcoreTools`. Indeed, the results may be sensitive to the designed tasks. It would be interesting to characterize the kinds of tasks performed with Kompren, which have a positive or negative impact (*e.g.* Task T_3) on different criteria including time, correctness and navigation effort.

Regarding the population validity, we asked the subjects to fill a questionnaire at the end of the experiments on their knowledge in modeling. We selected the data of subjects having good knowledge in software modeling. This selection permitted to analyze the data corresponding to a representative population of modeling practitioners.

Construct validity. This threat relates to the perceived overall validity of the experiments. Two threats may have affect the validity of our experiments: the learning gap and the tiredness. Unlike `Explen` and RAM, several subjects knew Eclipse, `EcoreTools`, and UML. To reduce this gap, we provided a training period on `Explen` and `EcoreTools` with another metamodel than RAM and UML. We did not provide a training period on the UML metamodel because, although it is not necessarily known by all modelers, the UML metamodel is still self-descriptive and uses common well-known elements. As for the tiredness, we limited the number of tasks per subjects and make sure that they can be performed in a reasonable time frame by evaluating their duration in the early experiments.

6. Related Work

6.1. Visualizing Models

Musial *et al.* applied focus+context techniques on UML models [30]. They proposed a lens showing different levels of detail of UML models. The level of detail of a UML element is computed according to the degree of interest in relation with the current situation. For instance, this lens reduces the details of

classes having little connections with a given one. We use this principle to build the Explen semantic zoom where the roles and cardinality of references may not be visible depending on the zoom level. This principle could be also used with Explen instead of hidden the metamodel elements.

Adora is a modeling tool that embeds hierarchical navigation features [31]. These features consist of semantic zooms that permit to select the desired level of detail. Users can also filter out individual nodes to reduce the size and complexity of diagrams without modifying the underlying model, as in EcoreTools.

Various works have been conducted on layout to propose new algorithms and methods for minimizing relations crossing [9, 10, 11]. Different guidelines for drawing class diagrams have also been proposed [12, 13]. Moreover, several research works proposed to represent class models differently than using class diagrams [14] or in 3D [15, 16, 17]. In our work, however, we focus on how to produce interactive visualization features dedicated to metamodels rather than the rendering of metamodels. We also keep the focus on the class diagram representation promoted and widely-used within the MDE community.

Complementary to our work, techniques have been proposed to visualize large UML class diagrams based on focus+context techniques [32, 33].

Regarding MDE activities, research works have been proposed to ease the development of model visualizers and their layout [34], or to use gestures within modeling editors [35]. Our work follows this trend that aims at easing the development process of modeling editors and completing them with advanced interactive navigation features.

Lange *et al.* proposed several views to improve the understanding of UML models [36]. The authors proposed a view, called *context view*, that consists of all the model elements related to a specific one. Such a view corresponds to slicing filter we proposed. However, our slicing filter provides parameters dedicated to metamodels (cardinality, radius, composition). Moreover, a contribution of our work is the use of Kompren to ease the development of such interactive navigation features. In the same work, Lange *et al.* also proposed a *metric view* that combines the rendering of class diagrams with various metrics. We think that such a view may improve metamodeling tasks related to metamodel quality (T_3 in our experiments).

Kagdi *et al.* [32] propose the use of onion graphs [37] as another focus+context technique for visualizing large UML class models. The focus area is presented in detail with standard UML notation while the rest of the model is abstracted at various levels of detail and presented in onion notation. The pure-onion notation represents abstractions in which a set of structural properties holds for all the members in the group (*e.g.* all the grouped classes have a generalization relationship). The onion supports semantic zooming and incremental exploration.

SDViz is an interactive system for visualizing technical diagrams [38]. This system embeds several interactive visualization features, in particular focus+context techniques, to keep contextual information while visualizing diagrams. *SDViz* does not provide users with interactive filtering features as those we proposed.

Ducasse *et al.* propose a visualization technique for understanding relations between packages [39]. Our work does not consider packages since we demonstrated that their use within metamodels is limited in practices.

In the context of MDE, the interest for the generation of graphical modeling editors from various description models [40] has been mainly emphasized by the Graphical Modeling Framework (GMF)[5] of the Eclipse project. *GMF* provides the infrastructure and components for the generation of graphical modeling editors. *GMF* relies on the definition a set of models that describe the modeling editor. These models are then compiled into Java code dedicated to run on top of the Eclipse platform. Still, *GEF3D* is a graphical framework for developing graphical 2D editors running on top of Eclipse [41]. Following the trend of *GMF*, various approaches have been proposed for modeling and generating modeling editors [42, 43, 44, 45][6]. To our best knowledge, neither of these approaches, *DaisyViz* [45] excluded, consider the modeling, the development, or the generation of interactive visualization features as we proposed using Kompren. *DaisyViz* is a a model-based user interface toolkit for developing domain-specific information visualization systems [45]. It allows programmers to develop visualization systems having advanced interactive visualization features such dynamic queries or focus+context. We think that our Explen prototype could have been developed using *DaisyViz* but no implementation is available.

Complementary to our work, several model-driven approaches focus on improving the interactivity while editing models graphically, for instance using sketching-based techniques [46, 47].

SHriMP is a visualization technique to browse and explore complex information spaces efficiently [48]. Initially designed for code comprehension, *SHriMP* has been generalized to flow diagrams. In addition to its semantic zoom, *SHriMP* provides a hierarchical overview of nodes of interest. Nodes can be filtered manually or following a given node type. This feature is close to our filters with the difference that ours are based on the specific characteristics of metamodels (inheritance, reference, cardinality, *etc.*) and not only the node type. Kompren can also permit developers of visualization tools to build domain-specific filters.

6.2. Interactive visualization techniques for graphs

Visualizing and interacting with graphs has been widely studied [49, 50, 51, 52, 53]. In this section, we will focus on interactive navigation features related to those we proposed.

PDQ Tree-browser is a graph visualization tool supplemented with several visualization techniques [54] This tool provides filtering features, that can be viewed as a query language, to reduce the data set to a smaller size.

Mondrian is a visualization framework for producing data views from scripts [55]. Such scripts can be considered as queries on the data under study. In our work we focus on interacting directly on the representation of metamodels rather than building view from data using a query language.

[5] http://www.eclipse.org/modeling/gmp/
[6] http://www.eclipse.org/sirius/

SpaceTree is a 2D visualization techniques for exploring large trees [56]. *SpaceTree* is generalized to trees and, therefore, provides not provide features tailed for metamodels or class diagrams. However, *SpaceTree* provides a filtering feature to remove the element irrelevant regarding the searched word. It also allows opening and closing tree branches as in tree views. This feature permits to manually reduce the amount of displayed data and could be added to `Explen`.

Closely related, visualization techniques dedicated to clustered graphs have been proposed [57]. The goal of these techniques is to alleviate the visualization of graphs thanks to a three dimensional representation sliced in layers. In a complementary manner, navigating throughout graphs using identified clusters can ease their understanding [58]. In our context, such approaches can be used to visualize highly nested models.

The use of lenses for visualizing graphs has also been proposed [59]. If our work does not focus on lenses, this visualization techniques should be investigated. For instance, Tominski *et al.* developed a lens dedicated for visualizing edges of local nodes by hiding the other edges [59].

6.3. Empirical Studies

Empirical studies have been conducted on the understanding of UML class diagrams when doing maintenance activities on class diagrams [60], or depending on the layout used [61, 62]. In particular, Purchase *et al.* highlight that when visualizing domain-specific graphs (*e.g.* class diagrams) the semantic of the domain should be considered in the layout process [62]. Our work follows this conclusion with the difference that interactive visualization techniques should be considered similarly.

Nugroho focuses on the level of detail of UML models and its impact on the impact on their comprehension [63]. This work exhibits that UML models with a higher level of detail improves their comprehension. However, the UML class diagrams used in the experiments were composed of 20 classes only. In `Explen`, the level of detail of a metamodel can be customized using the semantic zoom to permit users to select the level they expect.

The empirical study conducted by Lemon *et al.* on diagram comprehension highlights interesting results [64]. The results of this study shown that the size of diagrams (number of entities and relations), the relations crossing, and the number of bends per relations have a negative effect on diagram comprehension. These results motivate our work on reducing the displayed metamodel elements according to the users interest.

Störrle conducted several experiments on the effects of the layout of UML models on their comprehension [65]. The resulting conclusions state that good layout helps UML modelers to understand the model under study, even more for novice modelers. We paid only little attention to the layout we developed for `Explen`. So, the results in favor of `Explen` exhibited by our experiments may be improved thanks to a optimized layout.

Guéhéneuc developed a method based on eye-tracking for evaluating class diagrams comprehension [6]. This principle has also been used to measure: the efficiency of several design pattern representations using the UML class diagram notation [8]; the impact of the status and expertise of subjects when performing maintenance tasks on UML class diagrams [60]. Closely, Yusuf *et al.* studied the comprehension of UML class diagrams via eye tracking [7]. These works focus on class diagram representation and layout while our work targets interactive navigation features for metamodel. However, the use of eye-trackers may be used for validating the cognitive-load of modelers while interacting with metamodel editors.

6.4. Program and Model Slicing

Program slicing [20] is a "technique for focusing on certain aspects of a program's behavior and removing all other parts of code not concerned with this behavior [66]". Program slicing is an operation that takes as input slicing criteria, *i.e.* variables and their position in the program to slice. This operation produces as output a slice composed of the statements that have (or may have) effects on the slicing criteria. The two major slicing methods are static and dynamic slicing. The static slicing operation does not execute or interpret the program so that the output slice may not be minimal. Dynamic slicing remedies this drawback by evaluating the programs' statements. The interested reader can refer to [67, 68, 69, 70, 71] for more details on program slicing.

As introduced in Section 3, model slicing is a model comprehension technique inspired by program slicing. Model slicing is used for various purposes and with various modeling languages. For instance, slicing state-based models has been widely tackled in the literature, in particular for minimizing models [72, 73]. Another use of model slicing is the (meta-)model footprinting operation that statically or dynamically extracts elements from (meta-)models (*e.g.* from a model operation to get the effective metamodel used by the transformation) [74]. To our best knowledge, all these approaches produced as output a model (the output slice). Instead, we leverage the model slicing principles to build a filtering and viewpoint engine that can be mapped to metamodel visualization toolkits.

7. Conclusion and Future Work

7.1. Conclusion

Metamodels are cornerstones of MDE activities. Handling metamodels requires a good understanding of, or a part of, the metamodel under study. The current mainstream metamodel editors follow the graphical guidelines established for representing metamodels graphically in 2D. These editors, however, still rely on basic interactive features for navigating through and visualizing metamodels while advanced interactive navigation features exist. In this work, we conducted an empirical study to assess the benefits of several visualization techniques when performing different MDE activities. To do so, we implemented a metamodel visualizer, called `Explen`, we compared to the mainstream metamodel editor `EcoreTools`. The results of this study exhibits significant positive results for `Explen` regarding time, correctness, and navigation effort. The `Explen`'s visualization features were developed using `Kompren`, a domain specific language for modeling model slicers. We show that developing such interactive features can be eased when model slicing techniques are used.

7.2. Research Agenda

The studies described in this paper focus on metamodels and are a first step towards developing interactive navigation features for graphical model editors. More generally, models, that conform to their respective metamodels, can also be represented graphically using a dedicated textual or graphical concrete representation. Tools, such as `Sirius`[7], dedicated to the creation of modeling workbenches provide facilities for designing graphical concrete syntaxes and generating modeling editors. These tools, similarly to `EcoreTools`, still rely on basic visualization techniques. In future work, we will investigate the integration of the visualization techniques used in `Explen` into the development process of model editors. The goal is to provide these editors with dedicated visualization techniques to ease the visualization and navigation of any kind of models represented with a graphical syntax.

We will also investigate other kinds of interactive navigation features for graphs that could be applied to metamodels. For instance, the notion of dynamic inset consisting of painting visual insets for off-screen nodes should be studied [75]. Another improvement can be the support of animations when applying filters to preserve the mental map [76, 58].

The analysis of the results shows that the benefits of our proposal can be sensitive to the design of modeling tasks. A future work can focus on characterizing the kinds of tasks that benefit from sliced-based visualization features.

Finally, we also think that this approach can be adapted to UML class diagrams or code visualization. For instance, Sillito *et al.* conducted an experiment to identify questions programmers asked when doing software evolution tasks based on the source code graph [77]. The resulting questions have been classified into four categories: finding initial focus points; building on those points; understanding a subgraph; questions over groups of subgraphs. These categories are closed to the goal of the interactive visualization features we propose and applying our approach to this domain may be investigated.

Acknowledgements

This work is partially supported by the French BGLE Project CONNEXION and by NSERC (Natural Sciences and Engineering Research Council of Canada) research grants.

References

[1] OMG, UML 2.1.1 Specification (2007).

[2] S. Zhao, M. J. McGuffin, M. H. Chignell, Elastic hierarchies: Combining treemaps and node-link diagrams, in: Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization, 2005, p. 8.

[3] F. Fondement, P. Muller, L. Thiry, Big Metamodels Are Evil, in: Model Driven Engineering Languages and Systems, MODELS'13, 2013, pp. 138–153.

[4] M.-A. D. Storey, F. D. Fracchia, H. A. Müller, Cognitive design elements to support the construction of a mental model during software exploration, Journal of Systems and Software 44 (3) (1999) 171–185.

[7] `http://www.eclipse.org/sirius/`

[5] D. Gračanin, K. Matković, M. Eltoweissy, Software visualization, Innovations in Systems and Software Engineering 1 (2) (2005) 221–230.

[6] Y.-G. Guéhéneuc, TAUPE: towards understanding program comprehension, in: Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research, CASCON '06, 2006.

[7] S. Yusuf, H. Kagdi, J. Maletic, Assessing the comprehension of UML class diagrams via eye tracking, in: Program Comprehension, 2007. ICPC '07. 15th IEEE International Conference on, 2007, pp. 113–122.

[8] G. Cepeda Porras, Y.-G. Guéhéneuc, An empirical study on the efficiency of different design pattern representations in UML class diagrams, Empirical Softw. Eng. 15 (5) (2010) 493–522.

[9] J. Seemann, Extending the Sugiyama algorithm for drawing UML class diagrams: Towards automatic layout of object-oriented software diagrams, in: Graph Drawing, 1997, pp. 415–424.

[10] C. Gutwenger, M. Jünger, K. Klein, J. Kupke, S. Leipert, P. Mutzel, A new approach for visualizing UML class diagrams, in: Proc. of SoftVis'03, ACM Press, 2003, p. 179.

[11] K. Wong, D. Sun, On evaluating the layout of UML diagrams for program comprehension, Software Quality Journal 14 (2006) 233–259.

[12] H. Eichelberger, Nice class diagrams admit good design?, Proceedings of the 2003 ACM Symposium on Software Visualization (2003) 159–168.

[13] H. Eichelberger, K. Schmid, Guidelines on the aesthetic quality of UML class diagrams, Information and Software Technology 51 (12) (2009) 1686–1698.

[14] S. Ducasse, M. Lanza, The class blueprint: Visually supporting the understanding of classes, IEEE Transactions on Software Engineering 31 (2005) 75–90.

[15] M. Gogolla, O. Radfelder, M. Richters, Towards three-dimensional representation and animation of UML diagrams, in: Proc. of UML'99, 1999, pp. 1–12.

[16] T. Dwyer, Three dimensional UML using force directed layout, in: Proc. of APSEC'01, 2001, pp. 77–85.

[17] P. McIntosh, M. Hamilton, R. van Schyndel, X3D-UML: 3D UML state machine diagrams, in: ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems, 2008, pp. 264–279.

[18] A. Blouin, B. Combemale, B. Baudry, O. Beaudoux, Kompren: Modeling and generating model slicers, Software and Systems Modeling (SoSyM) (2012) 1–17.

[19] A. Blouin, B. Combemale, B. Baudry, O. Beaudoux, Modeling model slicers, in: ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems (MODELS'11), 2011, pp. 62–76.

[20] M. Weiser, Program slicing, in: Proceedings of the 5th international conference on Software engineering, IEEE Press, 1981, pp. 439–449.

[21] A. Blouin, N. Moha, B. Baudry, H. Sahraoui, Slicing-based techniques for visualizing large metamodels, in: IEEE Working Conference on Software Visualization (VISSOFT 2014), IEEE, 2014.

[22] C. Ware, H. Purchase, L. Colpoys, M. McGill, Cognitive measurements of graph aesthetics, Information Visualization 1 (2) (2002) 103–110.

[23] B. Shneiderman, Dynamic queries for visual information seeking, Software, IEEE (1994) 1–18.

[24] J.-M. Jézéquel, B. Combemale, O. Barais, M. Monperrus, F. Fouquet, Mashup of Meta-Languages and its Implementation in the Kermeta Language Workbench, Software and Systems Modeling.

[25] A. Blouin, O. Beaudoux, Improving modularity and usability of interactive systems with Malai, in: EICS'10: Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems, 2010, pp. 115–124.

[26] A. Blouin, B. Morin, O. Beaudoux, G. Nain, P. Albers, J.-M. Jézéquel, Combining Aspect-Oriented Modeling with Property-Based Reasoning to Improve User Interface Adaptation, in: EICS'11: Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems, 2011, pp. 85–94.

[27] J. Kienzle, W. Al Abed, F. Fleurey, J.-M. Jézéquel, J. Klein, Aspect-oriented design with reusable aspect models, Transactions on aspect-oriented software development (2010) 272–320.

[28] M. Fowler, Refactoring: improving the design of existing code, Addison-Wesley Professional, 1999.

[29] D. J. Sheskin, Handbook Of Parametric And Nonparametric Statistical Procedures, Fourth Edition, Chapman & Hall/CRC, 2007.

[30] B. Musial, T. Jacobs, Application of focus + context to UML, in: Proceedings of the Asia-Pacific symposium on Information visualisation - Volume 24, APVis '03, Australian Computer Society, 2003, pp. 75–80.

[31] M. Glinz, S. Berner, S. Joos, Object-oriented modeling with ADORA, Information Systems 27 (2002) 425–444.
[32] H. Kagdi, J. Maletic, Onion graphs for focus+context views of UML class diagrams, in: IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'07), 2007, pp. 80–87.
[33] M. Frisch, R. Dachselt, Off-screen visualization techniques for class diagrams, in: Proc. of SOFTVIS'10, 2010, p. 163.
[34] H. Fuhrmann, R. von Hanxleden, Taming graphical modeling, in: Proc. of MODELS'10, 2010, pp. 196–210.
[35] A. Scharf, T. Amma, Dynamic injection of sketching features into GEF based diagram editors, in: Proc. of ICSE'13, 2013, pp. 822–831.
[36] C. Lange, M. Chaudron, Interactive Views to Improve the Comprehension of UML Models - An Experimental Validation, in: 15th IEEE International Conference on Program Comprehension, 2007, pp. 221–230.
[37] G. Sindre, B. Gulla, H. G. Jokstad, Onion graphs: Aesthetics and layout, in: IEEE Workshop on Visual Languages, 1993, pp. 287–291.
[38] I. Woo, S. Y. Kim, R. Maciejewski, D. S. Ebert, T. D. Ropp, K. Thomas, SDViz: A Context-Preserving Interactive Visualization System for Technical Diagrams, in: Proceedings of the 11th Eurographics / IEEE - VGTC conference on Visualization, 2009, pp. 943–950.
[39] S. Ducasse, D. Pollet, M. Suen, H. Abdeen, I. Alloui, Package surface blueprints: Visually supporting the understanding of package relationships, in: Proc. of ICSM'07, 2007, pp. 94–103.
[40] M. Minas, G. Viehstaedt, DiaGen: a generator for diagram editors providing direct manipulation and execution of diagrams, in: Proceedings of Symposium on Visual Languages, IEEE, 1995, pp. 203–210.
[41] J. von Pilgrim, K. Duske, GEF3D: a framework for two-, two-and-a-half-, and three-dimensional graphical editors, in: Proceedings of the 4th ACM symposium on Software visualization, 2008, pp. 95–104.
[42] K. Ehrig, C. Ermel, S. Hänsgen, G. Taentzer, Generation of visual editors as eclipse plug-ins, in: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, 2005, p. 134.
[43] J. Grundy, J. Hosking, N. Zhu, N. Liu, N. Zealand, Generating Domain-Specific Visual Language Editors from High-level Tool Specifications, in: Proc. of ASE'06, 2006, pp. 25–36.
[44] R. I. Bull, M.-A. Storey, B. Columbia, J.-m. Favre, M. Litoiu, M. Ontario, An Architecture to Support Model Driven Software Visualization, in: Proc. of ICPC'06, 2006, pp. 100–106.
[45] L. Ren, F. Tian, X. (Luke) Zhang, L. Zhang, DaisyViz: A model-based user interface toolkit for interactive information visualization systems, Journal of Visual Languages & Computing 21 (4) (2010) 209–229.
[46] J. Grundy, J. Hosking, Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool, in: 29th International Conference on Software Engineering, 2007, pp. 21–26.
[47] W. A. Abed, V. Bonnet, M. Schöttle, TouchRAM: A multitouch-enabled tool for aspect-oriented software design, in: Software Language Engineering, 2013.
[48] D. Rayside, M. Litoiu, M.-A. Storey, Visualizing flow diagrams in websphere studio using SHriMP views, Information Systems Frontiers 5 (2) (2003) 161–174.
[49] I. Herman, G. Melançon, M. S. Marshall, Graph visualization and navigation in information visualization: A survey, IEEE Transactions on Visualization and Computer Graphics 6 (2000) 24–43.
[50] A. Cockburn, A. Karlson, B. B. Bederson, A review of overview+detail, zooming, and focus+context interfaces, ACM Comput. Surv. 41 (1) (2009) 2:1–2:31.
[51] P. Caserta, O. Zendra, Visualization of the Static Aspects of Software: A Survey, IEEE transactions on visualization and computer graphics 17 (7) (2010) 913–933.
[52] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. van Wijk, J.-D. Fekete, D. Fellner, Visual analysis of large graphs: State-of-the-art and future research challenges, Computer Graphics Forum 30 (6) (2011) 1719–1749.
[53] H. Schulz, S. Hadlak, H. Schumann, The design space of implicit hierarchy visualization: A survey, Visualization and Computer Graphics, IEEE Transactions on 17 (4) (2011) 393–411.
[54] H. P. Kumar, C. Plaisant, B. Shneiderman, Browsing hierarchical data with multi-level dynamic queries and pruning, International Journal of Human-Computer Studies 46 (1) (1997) 103 – 124.
[55] M. Meyer, T. Gîrba, M. Lungu, Mondrian: an agile information visualization framework, in: ACM symposium on Software visualization, 2006, pp. 135–144.
[56] C. Plaisant, J. Grosjean, B. Bederson, Spacetree: supporting exploration in large node link tree, design evolution and empirical evaluation, in: IEEE Symposium on Information Visualization, 2002, pp. 57–64.
[57] P. Eades, Q.-W. Feng, Multilevel visualization of clustered graphs, in: Graph drawing, 1997, pp. 101–112.
[58] P. Eades, M. L. Huang, Navigating clustered graphs using force-directed methods, J. Graph Algorithms Appl. 4 (3) (2000) 157–181.
[59] T. Christian, J. Abello, F. van Ham, H. Schumann, Fisheye tree views and lenses for graph visualization, in: Proceedings of the conference on Information Visualization, IV '06, 2006, pp. 17–24.
[60] Z. Soh, Z. Sharafi, B. Van den Plas, G. Porras, Y. Gueheneuc, G. Antoniol, Professional status and expertise for UML class diagram comprehension: An empirical study, in: Proc. of ICPC'12, 2012, pp. 163–172.
[61] B. Sharif, J. I. Maletic, The effect of layout on the comprehension of UML class diagrams: A controlled experiment, in: Proc. of Vissoft'09, 2009, pp. 11–18.
[62] H. Purchase, M. McGill, Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study, in: Proc. of APVIS'01, 2001, pp. 129–137.
[63] A. Nugroho, Level of detail in UML models and its impact on model comprehension: A controlled experiment, Information and Software Technology 51 (12) (2009) 1670–1685.
[64] K. Lemon, E. B. Allen, J. C. Carver, G. L. Bradshaw, An Empirical Study of the Effects of Gestalt Principles on Diagram Understandability, in: 1st International Symposium on Empirical Software Engineering and Measurement, IEEE, 2007, pp. 156–165.
[65] H. Störrle, On the Impact of Layout Quality to Understanding UML Diagrams: Diagram Type and Expertise, in: IEEE Symposium on Visual Languages and Human-Centric Computing, 2012, 2012, pp. 49–56.
[66] J. T. Lallchandani, R. Mall, A Dynamic Slicing Technique for UML Architectural Models, IEEE Transactions on Software Engineering 99.
[67] J. Silva, A vocabulary of program-slicing based techniques, ACM Computing Surveys.
[68] M. Harman, R. Hierons, An overview of program slicing, Software Focus 2 (3) (2001) 85–92.
[69] K. Gallagher, D. Binkley, Program slicing, in: In Proceedings of Frontiers of Software Maintenance, 2008.
[70] B. Xu, J. Qian, X. Zhang, Z. Wu, L. Chen, A brief survey of program slicing, SIGSOFT Softw. Eng. Notes 30 (2005) 1–36.
[71] F. Tip, A Survey of Program Slicing Techniques, Journal of Programming Languages 3 (1995) 121–189.
[72] B. Korel, I. Singh, L. Tahat, B. Vaysburg, Slicing of state-based models, in: Proc. of the IEEE International Conference on Software Maintenance (ICSM'03), 2003.
[73] K. Androutsopoulos, D. Binkley, D. Clark, N. Gold, M. Harman, K. Lano, Z. Li, Model projection: Simplifying models in response to restricting the environment, in: International Conference on Software Engineering (ICSE'11), 2011.
[74] C. Jeanneret, M. Glinz, B. Baudry, Estimating footprints of model operations, in: International Conference on Software Engineering (ICSE'11), 2011.
[75] S. Ghani, N. H. Riche, N. Elmqvist, Dynamic Insets for Context-Aware Graph Navigation, Computer Graphics Forum 30 (3) (2011) 861–870.
[76] P. Eades, W. Lai, K. Misue, K. Sugiyama, Preserving the mental map of a diagram, International Institute for Advanced Study of Social Information Science, Fujitsu Limited, 1991.
[77] J. Sillito, G. C. Murphy, K. De Volder, Questions programmers ask during software evolution tasks, in: Proc. of the 14th ACM SIGSOFT international symposium on Foundations of software engineering, 2006, p. 23.